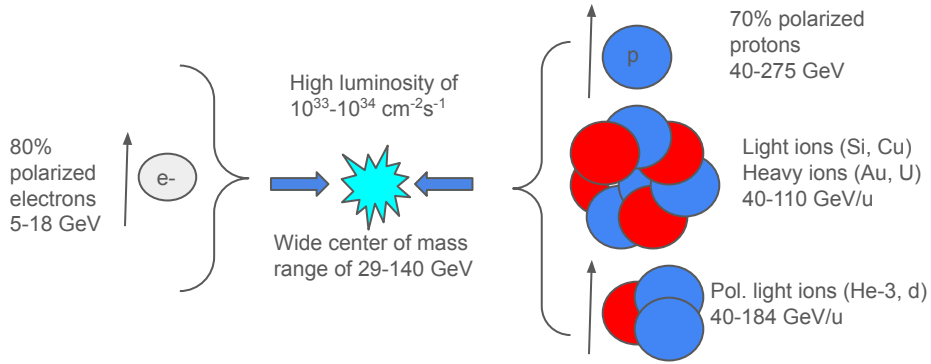
Machine Learning for ePIC Far Forward Detectors

Sakib Rahman (BNL), Alex Jentsch (BNL), and David Ruth (UNH)
April 3, 2025

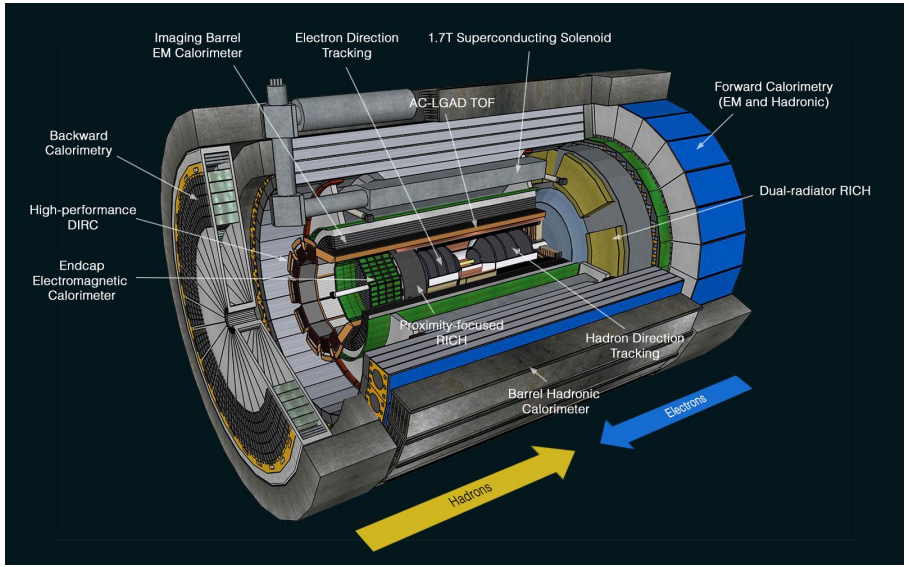
Introduction

ePIC, the first experiment at the future Electron-Ion Collider (EIC), will be realized in partnership of host labs - Brookhaven National Laboratory (BNL) and Jefferson Lab (JLab).

International collaboration with 173 institutions worldwide will provide insight into the nucleon and nucleus down to the scale of sea quarks and gluons leveraging unique and versatile EIC beam specs.



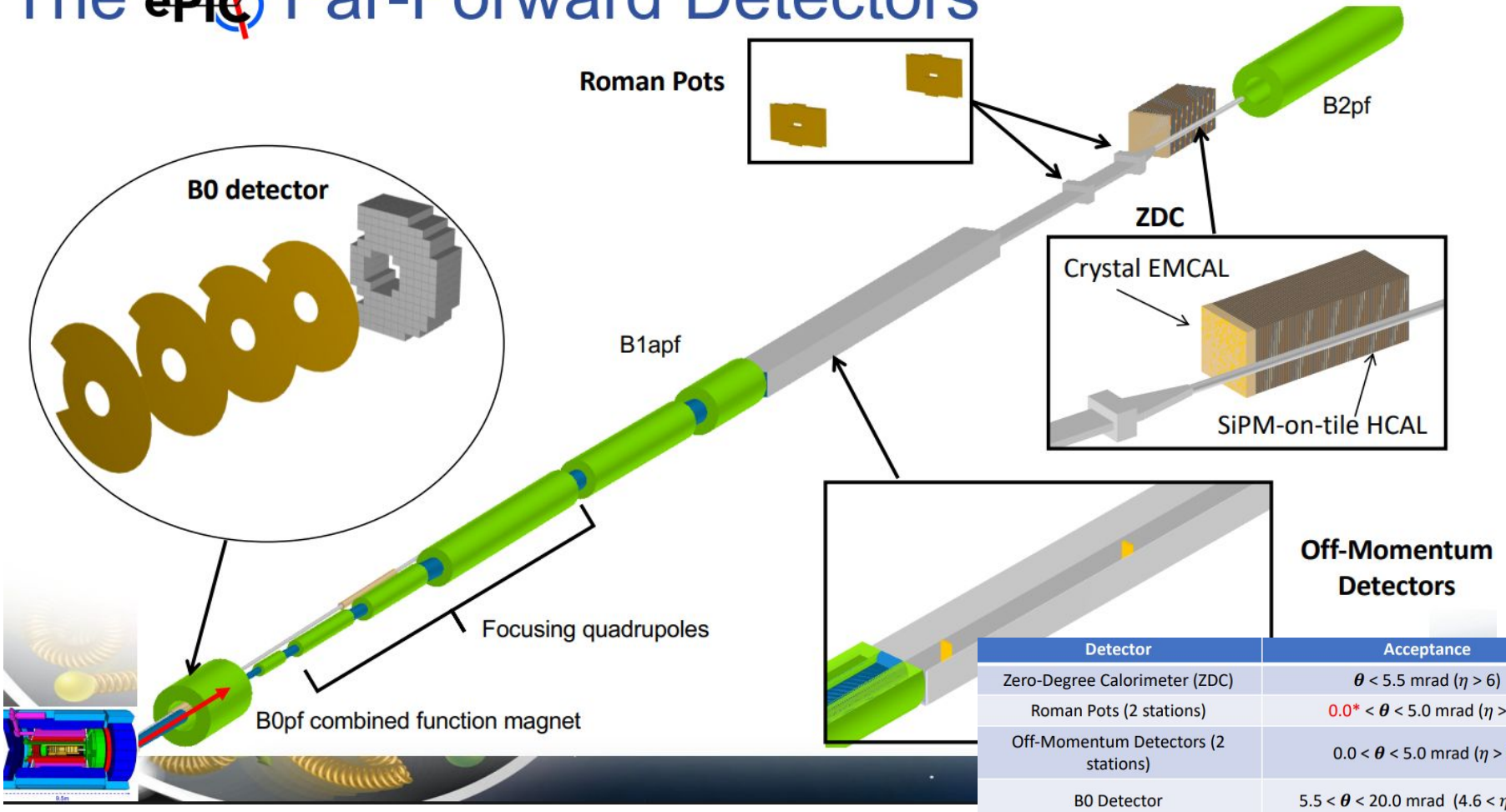
Highly integrated, multi-purpose detector to measure all final-state particles of interest with close to full acceptance and good resolution. Wide kinematic coverage provided by central detector + **far forward**/far backward systems.



This talk will focus on ML-based track reconstruction approach for the Roman Pots that can be generalized to other Far Forward Detectors relevant for exclusive interactions

Slide from A. Jentsch

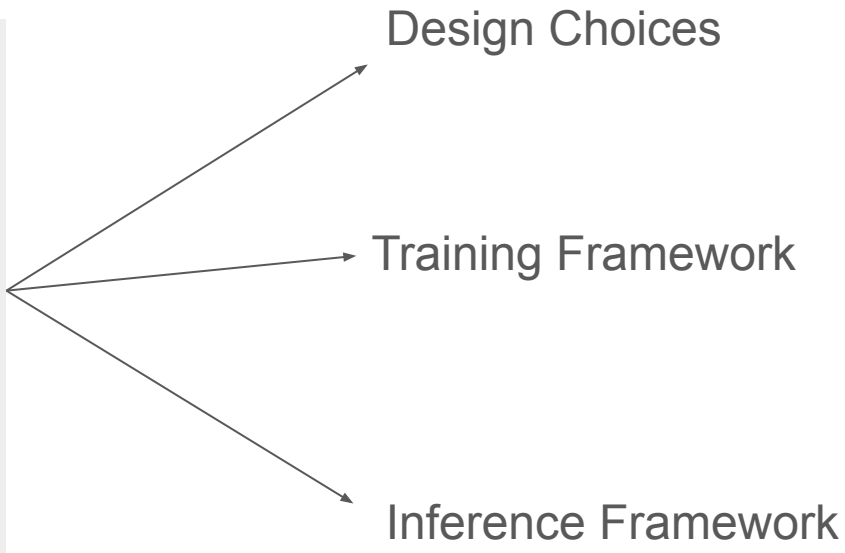
The ePIC Far-Forward Detectors



Detector	Acceptance
Zero-Degree Calorimeter (ZDC)	$\theta < 5.5$ mrad ($\eta > 6$)
Roman Pots (2 stations)	0.0* $< \theta < 5.0$ mrad ($\eta > 6$)
Off-Momentum Detectors (2 stations)	$0.0 < \theta < 5.0$ mrad ($\eta > 6$)
B0 Detector	$5.5 < \theta < 20.0$ mrad ($4.6 < \eta < 5.9$)

Motivation

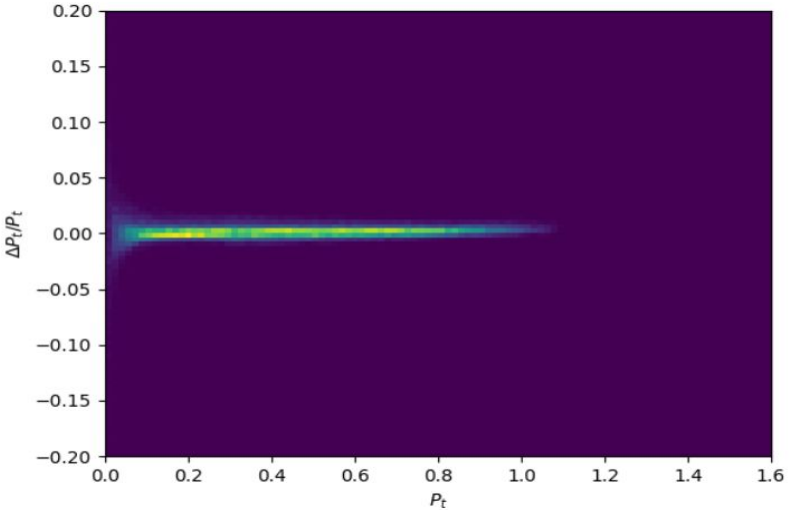
Aim is to develop a **machine learning training** and **inference** framework for momentum reconstruction for Roman Pots that can be integrated into our CI/CD infrastructure with proper benchmarking. Also satisfy version tracking and reproducibility requirements.



Design Choices

Work done by David Ruth (UNH) to identify independent multi-layer perceptron models (Px, Py, Pz) as the preferred option and a training data preprocessing script developed by Alex Jentsch (BNL) to convert hits from multiple layers into hit position and slope information (x, θ_x, y, θ_y).

Transverse momentum reconstruction demonstrates good resolution (<2%) except at very low values



- **Framework:** PyTorch
- **Architecture:** Multi-Layer Perceptron
- **5 Independent Models:**

$$\begin{pmatrix} x \\ \theta_x \\ y \\ \theta_y \end{pmatrix} \rightarrow (P_z)$$

$$\begin{pmatrix} x \\ \theta_x \\ P_z \end{pmatrix} \rightarrow (P_x)$$

$$\begin{pmatrix} y \\ \theta_y \\ P_z \end{pmatrix} \rightarrow (P_y)$$

$P_t < 0.3$

$$\begin{pmatrix} x \\ \theta_x \\ P_z \end{pmatrix} \rightarrow (P_x)$$

$P_t < 0.3$

$$\begin{pmatrix} y \\ \theta_y \\ P_z \end{pmatrix} \rightarrow (P_y)$$

- **5 Hidden Layers, 128 Neurons**
- **Loss Function:** Huber Loss
- **Optimizer:** Adam

Model trained with single particle primaries

Design Choices - Multilayer Perceptron

Neural Network

- Interconnected nodes called artificial neurons, organized into input, hidden and output layers
- Neurons in adjacent layers connected by weighted connections

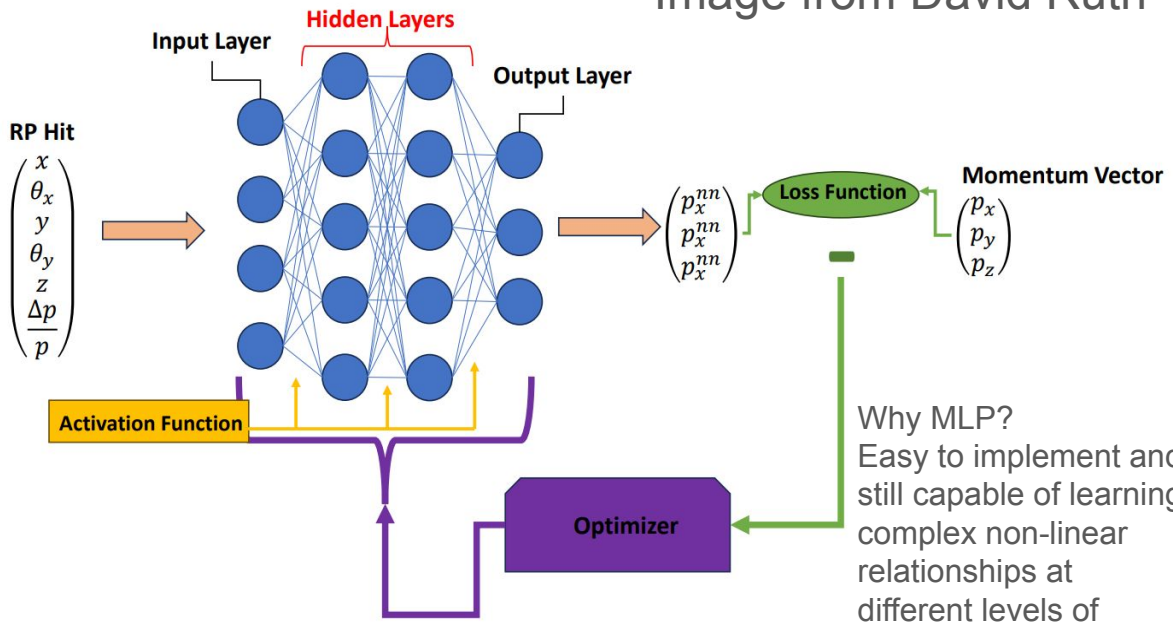
Properties of Multilayer Perceptron

- Dense/Fully Connected: Every neuron in a layer is connected to each of the neurons in subsequent layer
- Feedforward propagation: Activation function is applied to weighted sum of neurons going from one layer to the next.

$$y = f(z) = f\left(\sum_i (w_i \cdot x_i) + b\right)$$

Only in one direction (no loops)
Introduces non-linearity.

Image from David Ruth



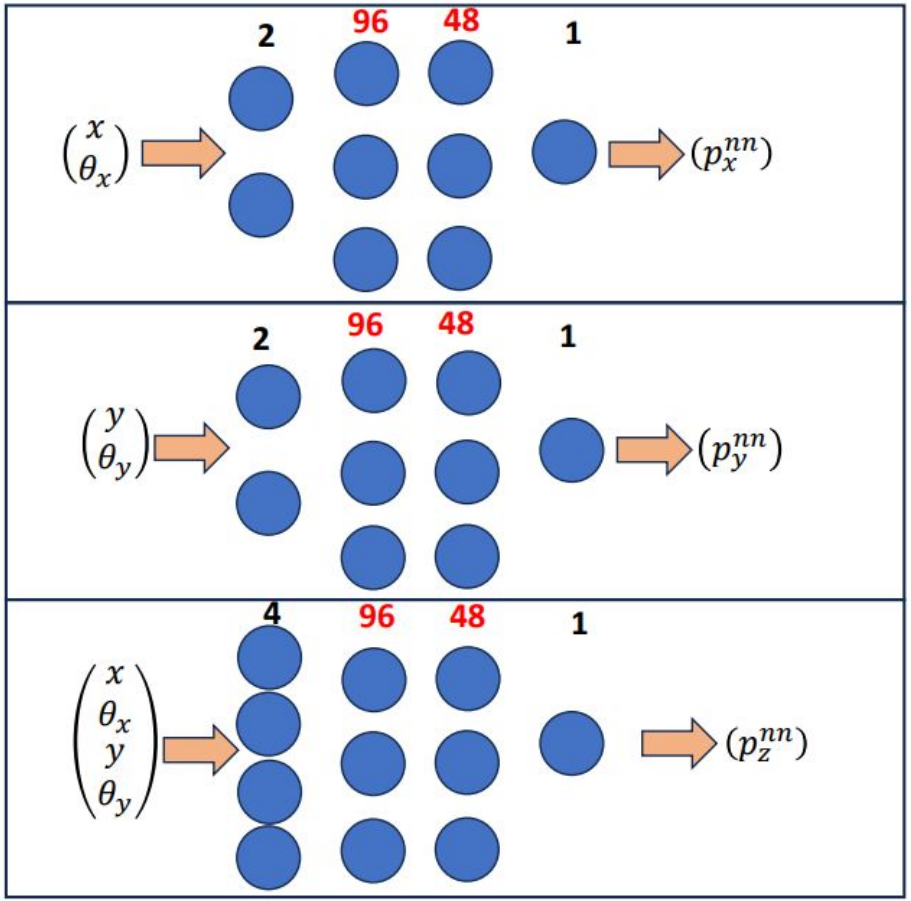
Why MLP?
Easy to implement and still capable of learning complex non-linear relationships at different levels of abstraction

- Back propagation: Loss function quantifying the difference between prediction and truth is “optimized” by adjusting the weights and biases in each layer using gradient descent.

Optimizers move the weights in small steps, adjusting the weights in the direction that decreases the error (loss). The size of the step is controlled by a parameter called the learning rate.

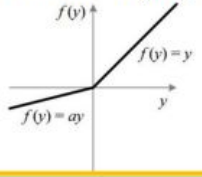
Design Choices

Slide from David Ruth



Activation Function: Leaky ReLU

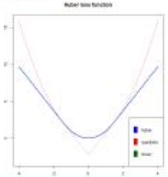
- ReLU: most popular activation function
- Adding a "leak" prevents the "dying neuron" problem



Loss Function: Huber Loss

- Takes outliers into account some but suppresses them
- Also tested with a custom loss function:

$$|2\tan^{-1}\frac{y}{x} - \frac{\pi}{2}|$$



Optimizer: Adam

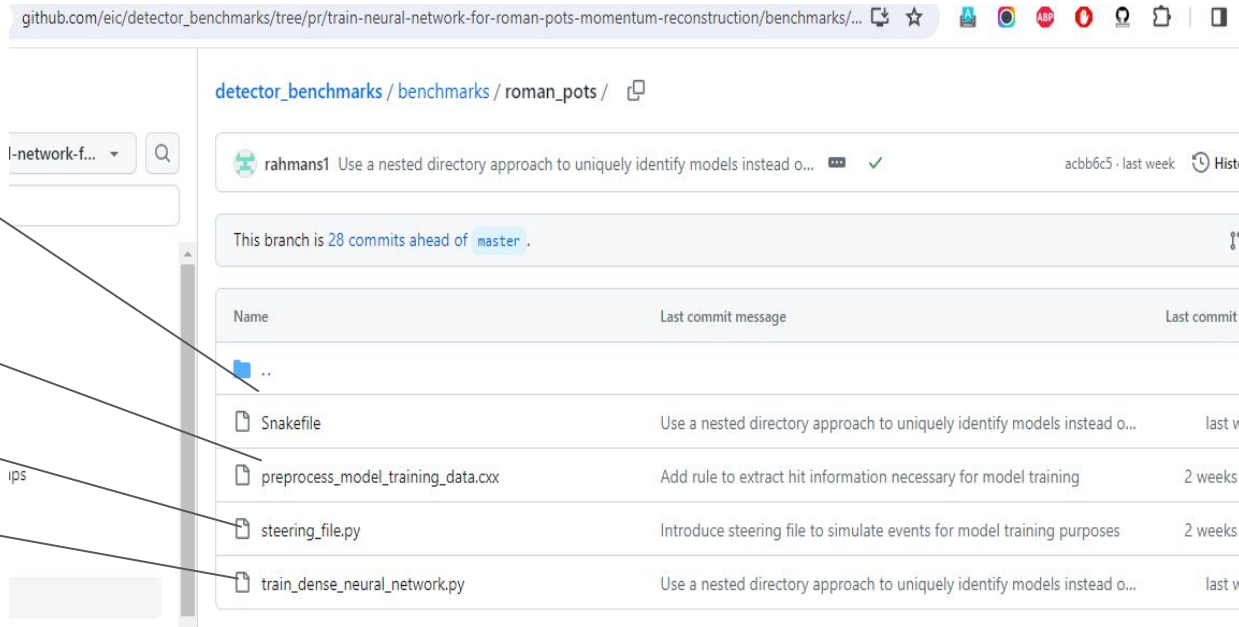
- Start learning rate at 0.01
- Employ a learning rate scheduler that halves LR on a plateau lasting 50 epochs or more

Training Framework

Chose to use snakemake for creating a model training procedure

- Workflow management system to create reproducible data analyses. Already the preferred choice for ePIC benchmarking analysis.
- Workflows can be seamlessly scaled to server, cluster, grid and cloud environments, without the need to modify the workflow definition.

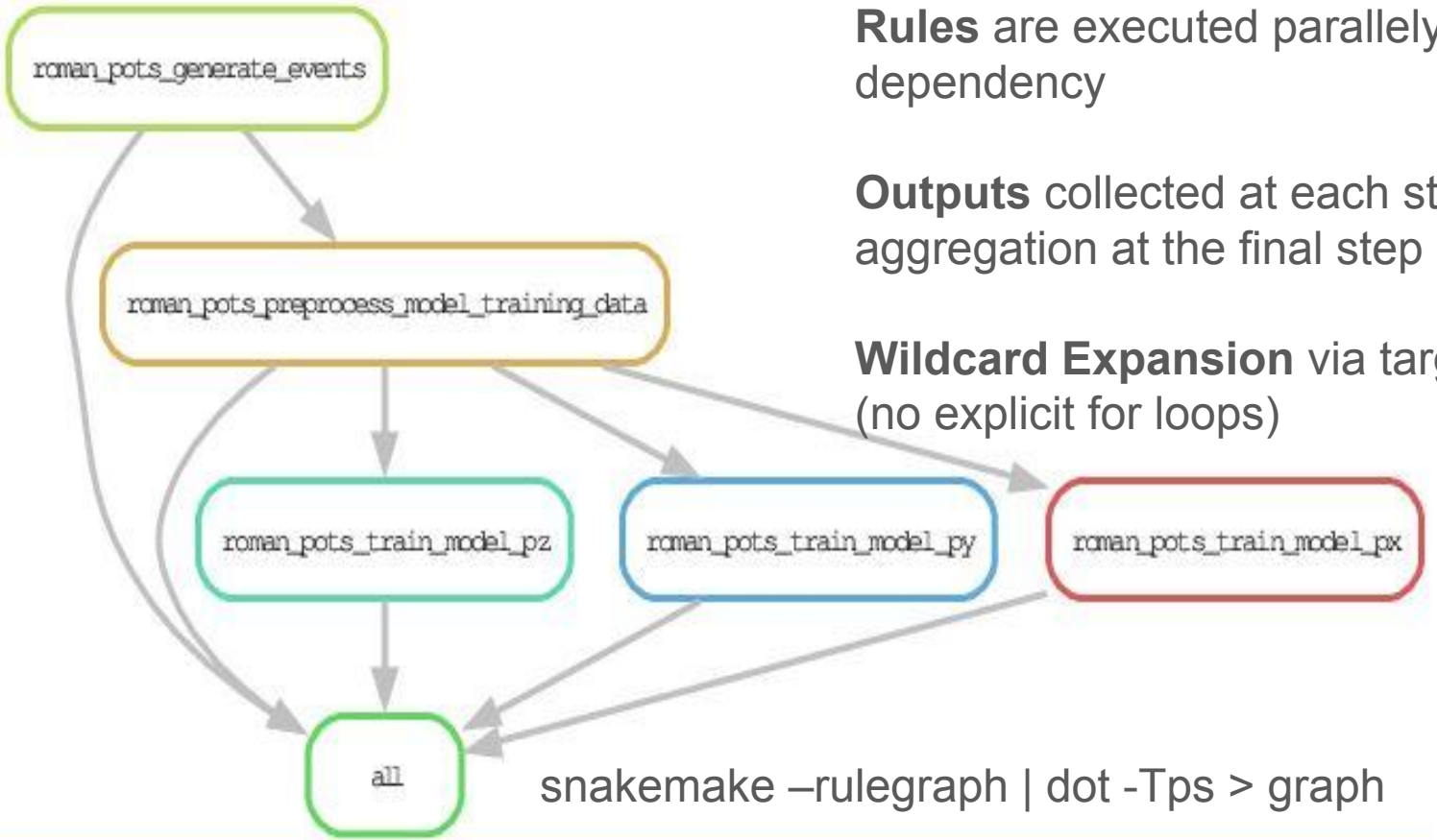
- Snakefile to combine everything into a workflow with proper dependency and parallelization
- Preprocessor script for training data
- DD4hep/Geant4 Simulation Config
- Parametrized model training script



Directory Structure

Training Framework - Snakefile

Visualize the snakemake workflow DAG



Rules are executed parallelly with proper dependency

Outputs collected at each step and aggregation at the final step

Wildcard Expansion via target rule “all” (no explicit for loops)

snakemake –rulegraph | dot -Tps > graph

Training Framework - Snakefile Breakdown

Global Variables and Hyperparameter Space for All Models

```
DETECTOR_PATH = os.environ["DETECTOR_PATH"]
DETECTOR_VERSION =
  re.search(r"epic-.*.-", DETECTOR_PATH).group().removeprefix("epic-").removesuffix("-")
SUBSYSTEM = "roman_pots"
BENCHMARK = "dense_neural_network"
DETECTOR_CONFIG = "epic_ip6"
NEVENTS_PER_FILE = 5
NFILES = range(1,6)
MODEL_PZ = {
  'num_epochs' : [100],
  'learning_rate' : [0.01],
  'size_input' : [4],
  'size_output': [1],
  'n_layers' : [3,6],
  'size_first_hidden_layer' : [128],
  'multiplier' : [0.5],
  'leak_rate' : [0.025],
}
```

Target Rule

```
rule all:
  input:

expand("results/"+DETECTOR_VERSION+"/"+DETECTOR_CONFIG+"/detec
tor_benchmarks/"+SUBSYSTEM+"/"+BENCHMARK+"/raw_data/"+DETECTOR
_VERSION+"_"+DETECTOR_CONFIG+"_{index}.edm4hep.root",
           index=NFILES),
```

Rule 1

```
rule roman_pots_generate_events:
    input:
        script="steering_file.py"
    params:
        detector_path=DETECTOR_PATH,
        nevents_per_file=NEVENTS_PER_FILE,
        detector_config=DETECTOR_CONFIG
    output:

"results/"+DETECTOR_VERSION+"/"+DETECTOR_CONFIG+"/detector_benchmarks/"+SUBSYSTEM+"/
"+BENCHMARK+"/raw_data/"+DETECTOR_VERSION+"_"+DETECTOR_CONFIG+"_{index}.edm4hep.root
"
    shell:
      """
      npsim --steeringFile {input.script} \
            --compactFile {params.detector_path}/{params.detector_config}.xml \
            --outputFile {output} \
            -N {params.nevents_per_file}
      """
```

To collect the outputs, wildcard expansion happens in target rule

Training Framework - Snakefile Breakdown

Extracts data from a rootfile and stores them in a text file with 7 columns that can be passed to the training scripts.

```
if (hasMCProton && hitLayerOne && hitLayerTwo) {

    double slope_x = (rpHitLayerTwo[0] - rpHitLayerOne[0]) / (rpHitLayerTwo[2] - rpHitLayerOne[2]);
    double slope_y = (rpHitLayerTwo[1] - rpHitLayerOne[1]) / (rpHitLayerTwo[2] - rpHitLayerOne[2]);

    outputTrainingFile << mcProtonMomentum[0] << "\t" << mcProtonMomentum[1] << "\t" << mcProtonMomentum[2] << "\t";
    outputTrainingFile << rpHitLayerTwo[0] << "\t" << slope_x << "\t" << rpHitLayerTwo[1] << "\t" << slope_y << endl;
}
```

Rule 2

Rule 1 Output Used As Input

Target Rule

```
rule roman_pots_preprocess_model_training_data:
  input:
    data =
"results/"+DETECTOR_VERSION+"/"+DETECTOR_CONFIG+"/detector_benchmarks/"+SUBSYSTEM+"/"+BENCHMA
RK+"/raw_data/"+DETECTOR_VERSION+"_"+DETECTOR_CONFIG+"_{index}.edm4hep.root",
    script = "preprocess_model_training_data.cxx"
  output:

"results/"+DETECTOR_VERSION+"/"+DETECTOR_CONFIG+"/detector_benchmarks/"+SUBSYSTEM+"/"+BENCHMA
RK+"/processed_data/"+DETECTOR_VERSION+"_"+DETECTOR_CONFIG+"_{index}.txt"
  shell:
    """
    root -q -b {input.script}\"(\\\"{input.data}\\\",\\\"{output}\\\")\"
    """
```

```
expand("results/"+DETECTOR_VERSION+"/"+DETECTOR_CONFIG+"/detector_benc
marks/"+SUBSYSTEM+"/"+BENCHMARK+"/processed_data/"+DETECTOR_VERSION+"_
+DETECTOR_CONFIG+"_{index}.txt",
           index=NFILES),
```

Wildcard expansion

Training Framework - Snakefile Breakdown

Rule 3

Rule 2 Output Used As Input

```
rule roman_pots_train_model_pz:
  input:
    data =
["results/"+DETECTOR_VERSION+"/"+DETECTOR_CONFIG+"/detector_benchmarks/"+SUBSYSTEM+"/"+BENCHM
ARK+"/processed_data/"+DETECTOR_VERSION+"_"+DETECTOR_CONFIG+"_{index}.txt".format(index=index
) for index in NFILES],
    script = "train_dense_neural_network.py"
  params:
    detector_version=DETECTOR_VERSION,
    detector_config=DETECTOR_CONFIG,
    subsystem=SUBSYSTEM,
    benchmark=BENCHMARK
  output:

"results/"+DETECTOR_VERSION+"/"+DETECTOR_CONFIG+"/detector_benchmarks/"+SUBSYSTEM+"/"+BENCHMA
RK+"/artifacts/model_pz/num_epochs_{num_epochs}/learning_rate_{learning_rate}/size_input_{siz
e_input}/size_output_{size_output}/n_layers_{n_layers}/size_first_hidden_layer_{size_first_hi
dden_layer}/multiplier_{multiplier}/leak_rate_{leak_rate}/model_pz.pt",

"results/"+DETECTOR_VERSION+"/"+DETECTOR_CONFIG+"/detector_benchmarks/"+SUBSYSTEM+"/"+BENCHMA
RK+"/artifacts/model_pz/num_epochs_{num_epochs}/learning_rate_{learning_rate}/size_input_{siz
e_input}/size_output_{size_output}/n_layers_{n_layers}/size_first_hidden_layer_{size_first_hi
dden_layer}/multiplier_{multiplier}/leak_rate_{leak_rate}/LossVsEpoch_model_pz.png"
  shell:
    """
    python {input.script} --input_files {input.data} --model_name model_pz --model_dir
results/{params.detector_version}/{params.detector_config}/detector_benchmarks/{params.subsys
tem}/{params.benchmark}/artifacts/model_pz/num_epochs_{wildcards.num_epochs}/learning_rate_{w
ildcards.learning_rate}/size_input_{wildcards.size_input}/size_output_{wildcards.size_output}
/n_layers_{wildcards.n_layers}/size_first_hidden_layer_{wildcards.size_first_hidden_layer}/mu
ltiplier_{wildcards.multiplier}/leak_rate_{wildcards.leak_rate} --num_epochs
{wildcards.num_epochs} --learning_rate {wildcards.learning_rate} --size_input
{wildcards.size_input} --size_output {wildcards.size_output} --n_layers {wildcards.n_layers}
--size_first_hidden_layer {wildcards.size_first_hidden_layer} --multiplier
{wildcards.multiplier} --leak_rate {wildcards.leak_rate}
    """
```

Target Rule

```
expand("results/"+DETECTOR_VERSION+"/{detector_config}/detector_benc
hmarks/"+SUBSYSTEM+"/"+BENCHMARK+"/artifacts/model_pz/num_epochs_{nu
m_epochs}/learning_rate_{learning_rate}/size_input_{size_input}/size
_output_{size_output}/n_layers_{n_layers}/size_first_hidden_layer_{s
ize_first_hidden_layer}/multiplier_{multiplier}/leak_rate_{leak_rate
}/model_pz.pt",
           detector_config=DETECTOR_CONFIG,
           num_epochs=MODEL_PZ["num_epochs"],
           learning_rate=MODEL_PZ["learning_rate"],
           size_input=MODEL_PZ["size_input"],
           size_output=MODEL_PZ["size_output"],
           n_layers=MODEL_PZ["n_layers"],

size_first_hidden_layer=MODEL_PZ["size_first_hidden_layer"],
           multiplier=MODEL_PZ["multiplier"],
           leak_rate=MODEL_PZ["leak_rate"]
           ),

expand("results/"+DETECTOR_VERSION+"/{detector_config}/detector_benc
hmarks/"+SUBSYSTEM+"/"+BENCHMARK+"/artifacts/model_pz/num_epochs_{nu
m_epochs}/learning_rate_{learning_rate}/size_input_{size_input}/size
_output_{size_output}/n_layers_{n_layers}/size_first_hidden_layer_{s
ize_first_hidden_layer}/multiplier_{multiplier}/leak_rate_{leak_rate
}/LossVsEpoch_model_pz.png",
           detector_config=DETECTOR_CONFIG,
           num_epochs=MODEL_PZ["num_epochs"],
           learning_rate=MODEL_PZ["learning_rate"],
           size_input=MODEL_PZ["size_input"],
           size_output=MODEL_PZ["size_output"],
           n_layers=MODEL_PZ["n_layers"],

size_first_hidden_layer=MODEL_PZ["size_first_hidden_layer"],
           multiplier=MODEL_PZ["multiplier"],
           leak_rate=MODEL_PZ["leak_rate"]
           ),
```

Wildcard expansion

Training Framework - Training Script (Run in Rule 3 and above)

Runs training given a set of hyperparameters. Model-agnostic, so it can run parallelly for Px, Py and Pz. Depending on model name (model_pz, model_py, model_px), picks the columns to use as input and target features.

(x,θ_x,y,θ_y) -> Pz, $(y,,\theta_{y,Pz})$->Py, (x,θ_x,Pz)->Px

```
    # Train models
    train_model(scaled_source, target, initial_model, hyperparameters)

    # Print end statement
    print("Training completed using "+str(len(hyperparameters.input_files))+" files with "+str(training_RP_pos_tensor.shape[0])+" eligible

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Train neural network model for roman pots")
    parser.add_argument('--input_files', type=str, nargs='+', required=True, help='Specify a location of input files.')
    parser.add_argument('--model_name', type=str, required=True, help='Specify model name.')
    parser.add_argument('--model_dir', type=str, required=True, help='Specify location to save model')
    parser.add_argument('--num_epochs', type=int, required=True, help='Specify number of epochs')
    parser.add_argument('--learning_rate', type=float, required=True, help='Specify learning rate')
    parser.add_argument('--size_input', type=int, required=True, help='Specify input size')
    parser.add_argument('--size_output', type=int, required=True, help='Specify output size')
    parser.add_argument('--n_layers', type=int, required=True, help='Specify number of layers')
    parser.add_argument('--size_first_hidden_layer', type=int, required=True, help='Size of first hidden layer')
    parser.add_argument('--multiplier', type=float, required=True, help='Specify mutilplier to calculate size of subsequent hidden layers')
    parser.add_argument('--leak_rate', type=float, required=True, help='Specify leak rate')
    hyperparameters = parser.parse_args()
    run_experiment(hyperparameters)
```

Training Framework - Usage and Result Structure

```
results
|____23.12.0
| |____epic_ip6
| | |____detector_benchmarks
| | | |____roman_pots
| | | | |____dense_neural_network
| | | | | |____artifacts
| | | | | | |____model_py
| | | | | | | |____num_epochs_100
| | | | | | | | |____learning_rate_0.01
| | | | | | | | | |____size_input_3
| | | | | | | | | | |____size_output_1
| | | | | | | | | | | |____n_layers_6
| | | | | | | | | | | | |____size_first_hidden_layer_128
| | | | | | | | | | | | | |____multiplier_0.5
| | | | | | | | | | | | | | |____leak_rate_0.025
| | | | | | | | | | | | | | | |____LossVsEpoch_model_py.png
| | | | | | | | | | | | | | | |____model_py.pt
| | | | | | | | | | | |____n_layers_3
| | | | | | | | | | | | |____size_first_hidden_layer_128
| | | | | | | | | | | | | |____multiplier_0.5
| | | | | | | | | | | | | | |____leak_rate_0.025
| | | | | | | | | | | | | | | |____LossVsEpoch_model_py.png
| | | | | | | | | | | | | | | |____model_py.pt
| | | | | | |____model_pz
| | | | | | | |____num_epochs_100
| | | | | | | | |____learning_rate_0.01
| | | | | | | | | |____size_input_4
| | | | | | | | | | |____size_output_1
| | | | | | | | | | | |____n_layers_6
| | | | | | | | | | | | |____size_first_hidden_layer_128
| | | | | | | | | | | | | |____multiplier_0.5
| | | | | | | | | | | | | | |____leak_rate_0.025
| | | | | | | | | | | | | | | |____LossVsEpoch_model_pz.png
| | | | | | | | | | | | | | | |____model_pz.pt
| | | | | | | | | | | |____n_layers_3
| | | | | | | | | | | | |____size_first_hidden_layer_128
| | | | | | | | | | | | | |____multiplier_0.5
| | | | | | | | | | | | | | |____leak_rate_0.025
| | | | | | | | | | | | | | | |____model_pz.pt
| | | | | | | | | | | | | | | |____LossVsEpoch_model_pz.png
| | | | | | |____model_px
| | | | | | | |____num_epochs_100
| | | | | | | | |____learning_rate_0.01
| | | | | | | | | |____size_input_3
| | | | | | | | | | |____size_output_1
| | | | | | | | | | | |____n_layers_7
| | | | | | | | | | | | |____size_first_hidden_layer_128
| | | | | | | | | | | | | |____multiplier_0.5
| | | | | | | | | | | | | | |____leak_rate_0.025
| | | | | | | | | | | | | | | |____LossVsEpoch_model_px.png
| | | | | | | | | | | | | | | |____model_px.pt
| | | | | | | | | | | |____n_layers_3
| | | | | | | | | | | | |____size_first_hidden_layer_128
| | | | | | | | | | | | | |____multiplier_0.5
| | | | | | | | | | | | | | |____leak_rate_0.025
| | | | | | | | | | | | | | | |____model_px.pt
| | | | | | | | | | | | | | | |____LossVsEpoch_model_px.png
| | | | |____raw_data
| | | | | |____23.12.0_epic_ip6_4.edm4hep.root
| | | | | |____23.12.0_epic_ip6_5.edm4hep.root
| | | | | |____23.12.0_epic_ip6_1.edm4hep.root
| | | | | |____23.12.0_epic_ip6_3.edm4hep.root
| | | | | |____23.12.0_epic_ip6_2.edm4hep.root
| | | | |____processed_data
| | | | | |____23.12.0_epic_ip6_1.txt
| | | | | |____23.12.0_epic_ip6_4.txt
| | | | | |____23.12.0_epic_ip6_3.txt
| | | | | |____23.12.0_epic_ip6_2.txt
| | | | | |____23.12.0_epic_ip6_5.txt
```

Run the workflow:

snakemake -c<ncores>

Can be submitted as part of a interactive job on a cluster or to batch system

Output is stored in a nested directory structure

Inference Framework

Not fully implemented for the Far Forward detectors yet but intend to leverage the common ONNX inference algorithm within EICrecon

Need an algorithm in eicrecon to extract data from the edm4hep output and process it into an input edm4eic tensor object and an algorithm to decipher the result tensor

npsim/geant4

edm4hep
data model

eicrecon

edm4eic
data model

Common ONNX inference algorithm

```
class ONNXInference : public ONNXInferenceAlgorithm,
                      public WithPodConfig<ONNXInferenceConfig> {

public:
  ONNXInference(std::string_view name)
      : ONNXInferenceAlgorithm{name,
                               {"inputTensors"},
                               {"outputTensors"},
                               ""} {
  }

  void init() final;
  void process(const Input&, const Output&) const final;

private:
  mutable Ort::Env m_env{nullptr};
  mutable Ort::Session m_session{nullptr};

  std::vector<std::string> m_input_names;
  std::vector<const char*> m_input_names_char;
  std::vector<std::vector<std::int64_t>> m_input_shapes;

  std::vector<std::string> m_output_names;
  std::vector<const char*> m_output_names_char;
  std::vector<std::vector<std::int64_t>> m_output_shapes;
};
```

Implemented by Dmitry Kalinkin

Inference Framework - What does the inference algorithm do?

Input Validation: Checks if the number of input tensors matches the expected number for the ONNX model. If not, it throws an error.

Input Tensor Preparation: The input tensors are prepared by converting edm4eic::Tensor objects into ONNX Runtime tensors (Ort::Value) using the iters_to_tensor helper function, handling both float and int64 types.

Shape Consistency Check: For each input tensor, it verifies that its shape matches the expected shape (using check_shape_consistency).

Inference: Runs the model using m_session.Run and catches any exceptions during inference. The output tensors are stored in onnx_values.

Output Processing: The outputs are processed by copying the data from ONNX tensors back into edm4eic::MutableTensor objects. The function handles different element types (float, int64) and constructs the output tensor accordingly.

Summary

- ePIC is the first experiment at the future Electron-Ion Collider (EIC)
- Highly integrated, multi-purpose detector to measure all final-state particles of interest with close to full acceptance and good resolution. Wide kinematic coverage provided by central detector + far forward/far backward systems.
- Far forward detectors (Roman Pots, off-momentum detectors, etc.) are critical for understanding exclusive interactions at ePIC
- Developed a machine learning training and inference framework for momentum reconstruction for Roman Pots that can be integrated into our CI/CD infrastructure with proper benchmarking. Also satisfy version tracking and reproducibility requirements.
    - Design Choice: Multilayer Perceptron Models for each momentum component
    - Training Framework: Snakemake in Gitlab CI
    - Inference Framework: ONNX in ePIC Reconstruction Software (EICrecon) using PODIO objects