Title

Subtitle

Author 1, Author 2, Author 3,
Author 4, **Presenter**, Author 6

Collaboration

Workshop, November 29, 2021

The DeGeSim Project
HELMHOLTZAI
ARTIFICIAL INTELLIGENCE
COOPERATION UNIT

RWTH AACHEN UNIVERSITY

JÜLICH Forschungszentrum
JÜLICH SUPERCOMPUTING CENTRE

DESY.

Vision

Save CPU time by using a **Neural Network** to simulate the **HGCAL**.

Use Graph Neural Networks (‘GNNs’) to deal with sparsity and irregular geometry

Design Decisions

- Stay as close as possible to the detector geometry $\Rightarrow$ Node $\equiv$ Cell
- Q1: When is the neighborhood between the cells constructed?
  - 'Post simulation' step: construct the neighborhood for each event (in CMSSW)
    - simulation always matches geometry

  vs.
  - Preprocessing step: read neighborhood from lookup table
    - faster $\Leftarrow$
- Q2: How do we continue after the simulation?

Simulation and Machine Learning Frameworks

Integration: Simulate in CMSSW, use the integrated ML software stack.

- Directly integrated in the production process

Decoupling: Simulate in CMSSW, continue with standard ML software stack.

- Lean development environment
- Faster development cycle
- Easier onboarding
- Access to HPC clusters w/o CMS software available
- Cutting edge software versions

$\Rightarrow$ Preferred solution for this case

The integrated ML tool chain in CMSSW is also widely used in CMS!

Loading the dataset

1. Read simulated hits from ROOT file
   - Most tools cannot handle data of variable dimension
   - `uproot` → `awkward` arrays
2. Convert simulated hits to graphs
   - Select the active cells from the extracted geometry
   - Extract the cell properties, construct the neighborhood information

     CPU intensive ⇒ parallel processing
3. Batch the graphs (`torch_geometric` [1])
4. Move to GPU

Custom dataloader based on `torch.multiprocessing` [2]

Loading the dataset

1. Read Simulated hits from ROOT file
   - Most tools cannot handle data of variable dimension
   - `uproot` → `akward` arrays
2. Convert simulated hits to graphs
   - Select the active cells from the extracted geometry
   - Extract the cell properties, construct the neighborhood information
     CPU intensive ⇒ parallel processing
3. Batch the graphs (`torch_geometric` [1])
4. (De)Serialize files (from) to disc (`torch.save`)
5. Move to GPU

Custom dataloader based on `torch.multiprocessing` [2]

Thank you!

DESY. Deutsches
Elektronen-Synchrotron
www.desy.de

Author 1, Author 2, Author 3,
Author 4, **Presenter**, Author 6
CMS-E
moritz.scham@desy.de

Bibliography I

[1] *Mini-batches*. URL: https://pytorch-geometric.readthedocs.io/en/latest/notes/introduction.html#mini-batches.

[2] *Multiprocessing package*. URL: https://pytorch.org/docs/stable/multiprocessing.html.