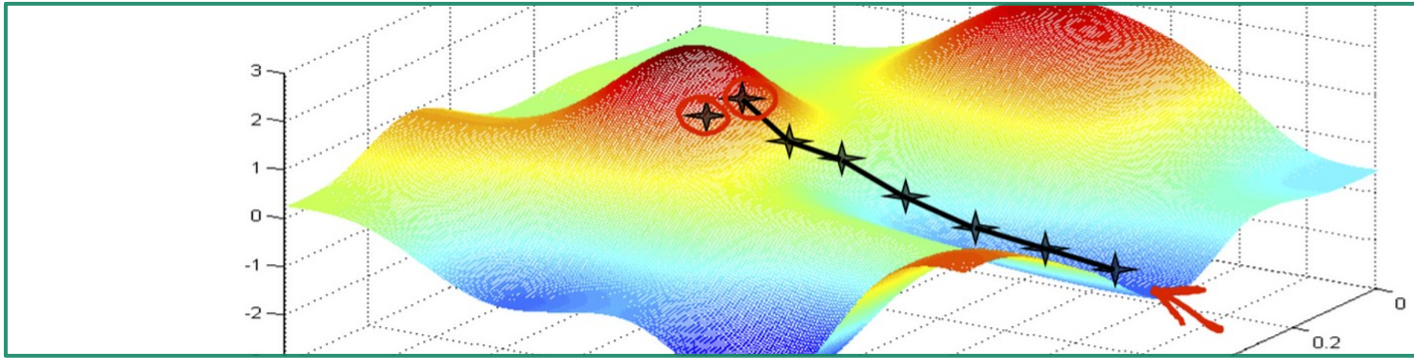
AIDO – Framework for AI Detector Optimization

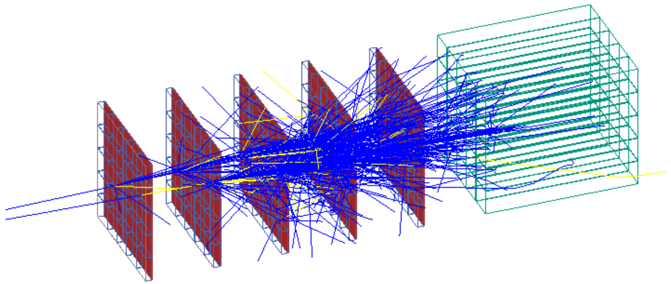
Kylian Schmidt, Jan Kieseler, Nikhil Krishna



[1]

Introduction

- Detectors are increasingly complex and specialized
- Competing criteria for what makes a good detector (physics reach, cost-effectiveness, size constraints...)
- On the other hand we have accurate geant4 simulations and ever-improving ML tools
- Can we translate the task of designing a good detector into an ***optimization problem***?



Motivation from my master thesis: what is the optimal distance between tracking layers for reconstructing ALPs?

The case for a Digital Twin approach

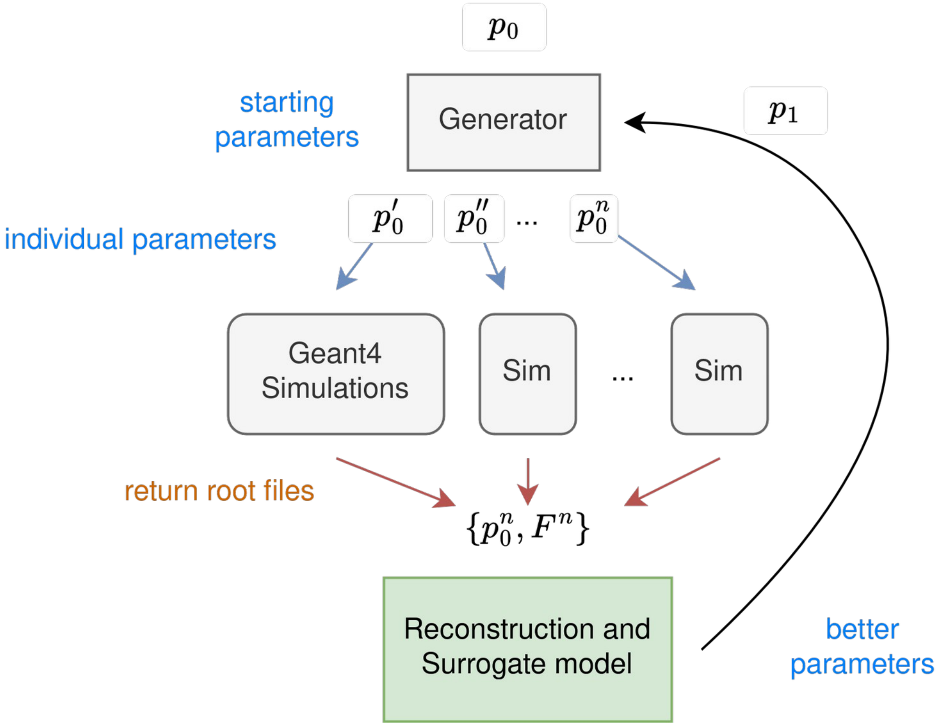
- A direct grid search would scale with O^N for N parameters

- Simple machine learning approach:
  - Simulate a given configuration, compute its performance and step forward along best gradient
  - Already an improvement over the grid search
  - But serialized and therefore extremely slow
  - Wasteful of resources since small changes in configuration are smooth

- Best solution: Construct a Digital Twin
  - A **Surrogate model** that behaves like the detector in a small region of parameter space and sample from it directly
  - Fast gradients computation based on this model
  - Trained on data simulated with parallel processing on HPC

Workflow of sampling and learning

- Sample inside a small region of parameter space
- Compute detector performance for each configuration
- Train the Surrogate model
- Optimize the parameters based on the Surrogate's prediction
- New predicted parameters are the mean of the next iteration
- Repeat until converged



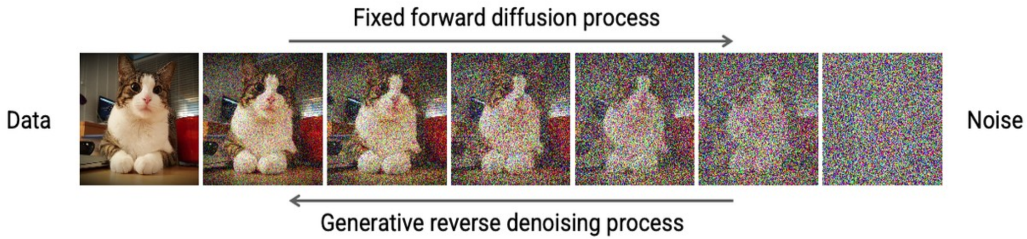
Surrogate (diffusion) model

- Interpolate the input data with a diffusion-like approach
- Predicts the output of the reconstruction model for new, unseen parameters

Principles of diffusion models:

Gradually add noise to the input and learn how much noise was added

Sample new results from random noise (and detector parameters)

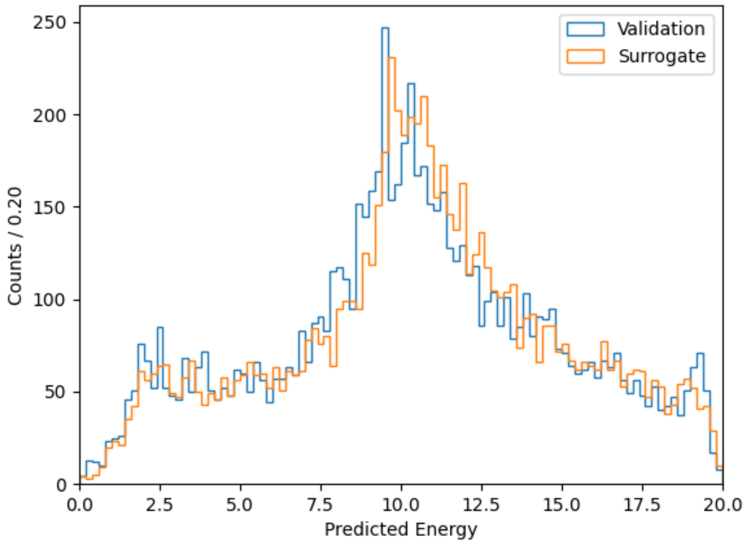


$$q(x_t|x_{t-1})$$

Model based on https://arxiv.org/pdf/2006.11239, Image from [2]

Surrogate (diffusion) model

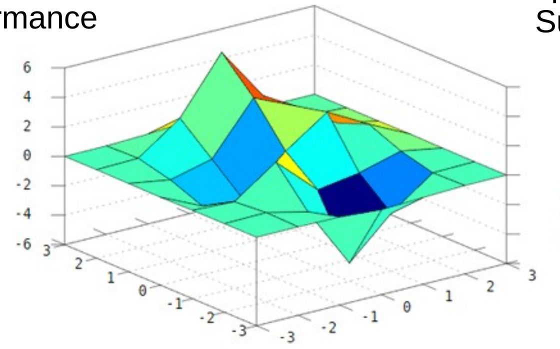
- **Training** provides:
  - Detector parameters
  - Context information
  - Targets (e.g. true initial energy)
  - Reconstruction results (predicted energy by the reconstruction algorithm)
- **Sample** after training using:
  - New, unseen detector parameters
  - Context
  - Targets
  - Outputs an approximation of the reconstruction results



Optimization on interpolated parameters

- Optimizer Loss computes gradients based on
  - The expected detector performance predicted by the Surrogate
  - User-defined constraints (cost, size, etc...)
  - A boundary loss that ensures that the parameters stay within the sampled region
- After Optimization, boost the parameters along the direction of change

Discretely sampled detector configurations with known performance



Interpolated detector space using the Surrogate

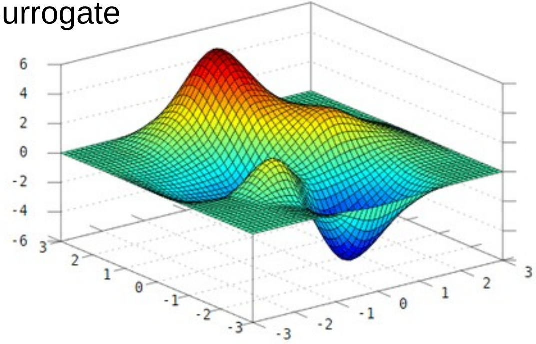
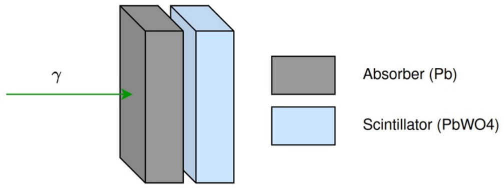


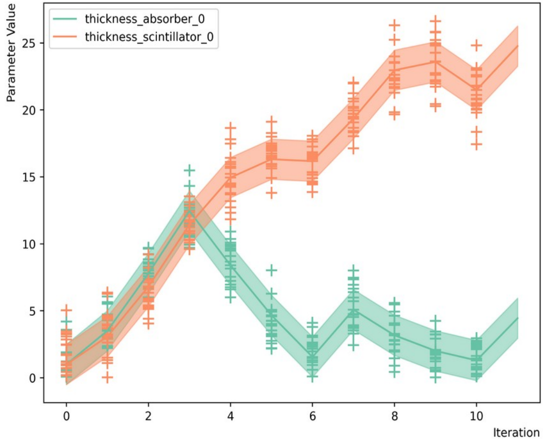
Image from [3]

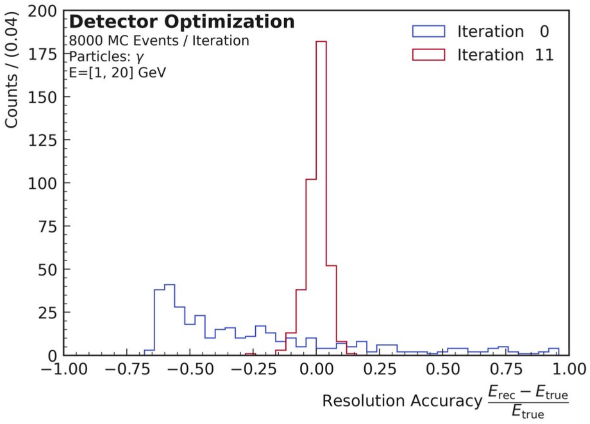
Sanity check on a simple setup

- Shoot 1-20 GeV photons
- Use a DNN to reconstructed the initial energy
- Goal: Maximize the energy resolution
- Confirms the expectation: large block of PbWO4



$$\mathrm{Loss} = \frac{(y^{\mathrm{pred}} - y)^2}{y + 1}$$

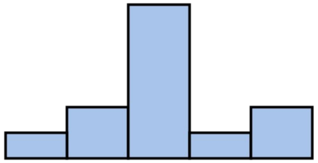




Now, discrete parameters

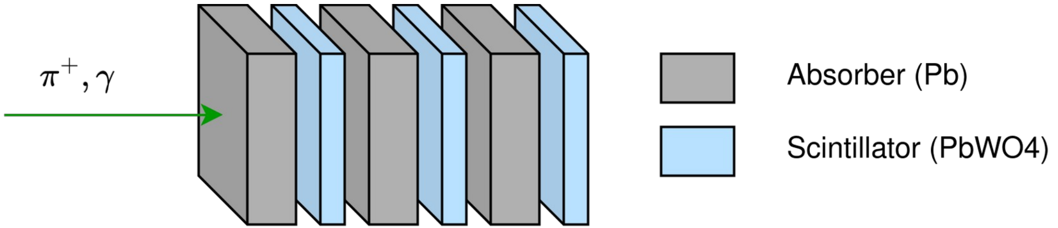
Institute of Experimental Particle Physics

- Always a challenge with ML due to discontinuities
- We use two representations:
  - Categories as probabilities that the optimizer can learn
  - One-hot encoding when sampling to have one hard value
- Surrogate model trains on one-hot but is able to interpret probabilities too (benefit of DNNs)
- Same idea as a classifier with softmax activation

Possible values	["A", "B", "C", "D", "E"]
Current value	"C"
One-Hot Encoding	$[0, 0, 1, 0, 0]$
Probabilities	 $[0.1, 0.2, 0.4, 0.1, 0.2]$

Example - Sampling Calorimeter

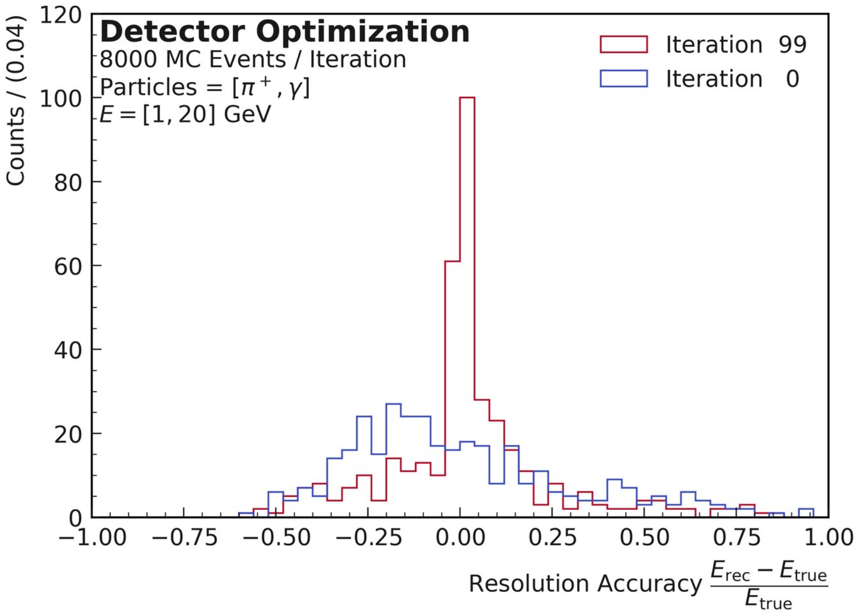
- Start from a random configuration and converge towards a usable design
- Shoot photons and charged pions with random energy
- Target: maximize the energy resolution

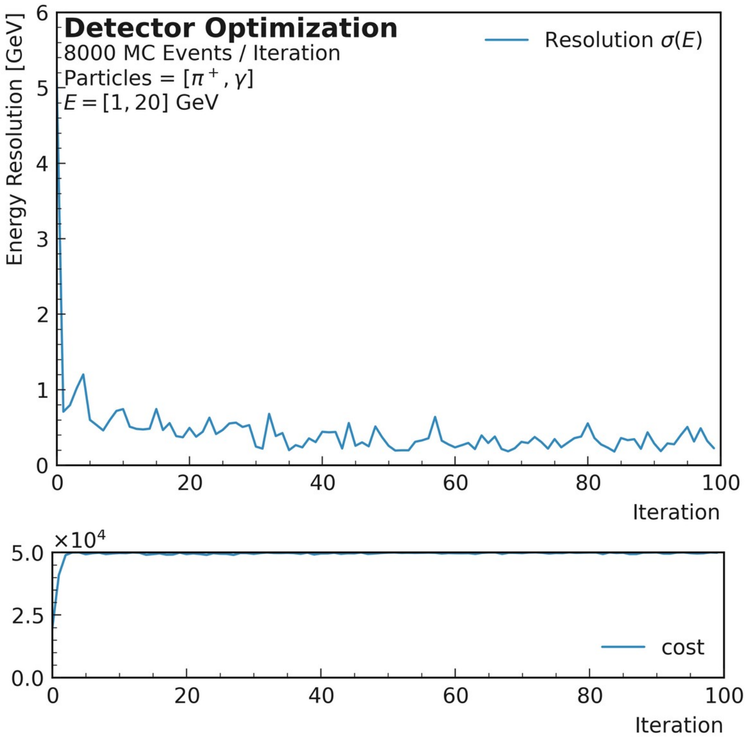


$$\mathrm{Loss} = \frac{(y^{\mathrm{pred}} - y)^2}{y+1}$$

- Include cost and size constraints
- Balance the material choice between performance and cost (expensive PbWO4 vs cheap Polystyrene)

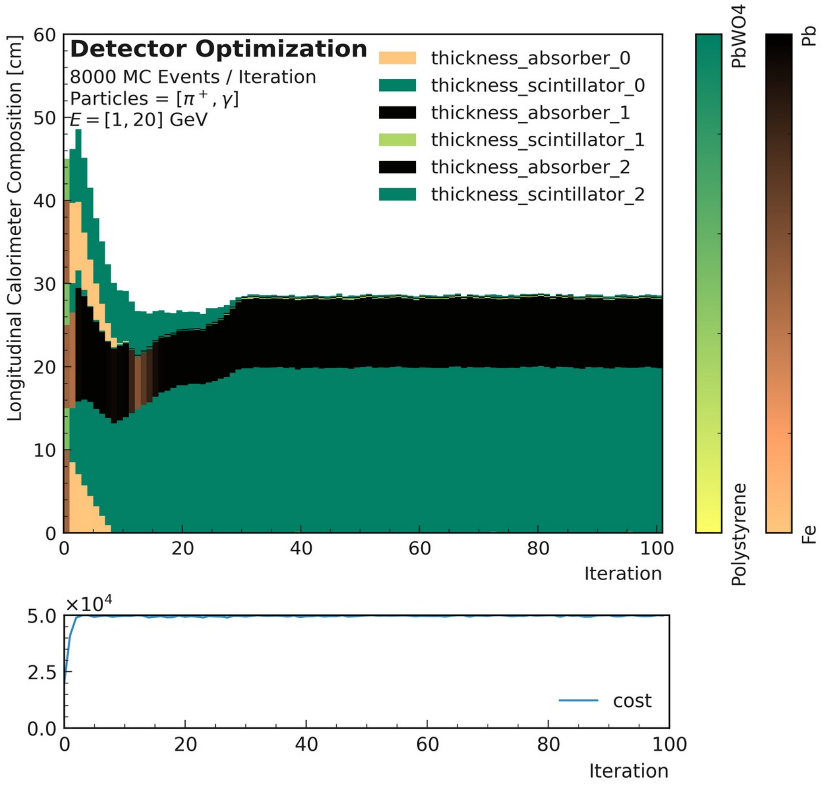
Example – Sampling calorimeter





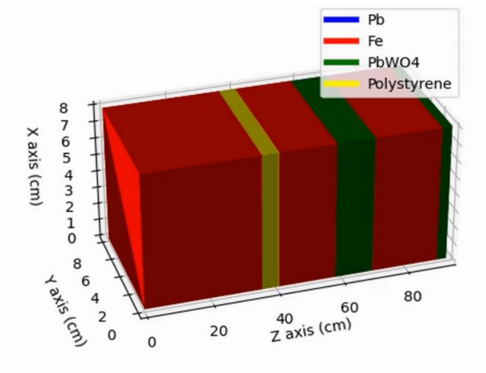
Example – Sampling Calorimeter

- Given our budget, the model proposes a large $PbWO_4$ detector with a Pb backplate
- Pretty good guess for an ECAL
- Pions however would benefit from a longer detector
- Having a pure ML task leads to ML specific challenges, namely local minima

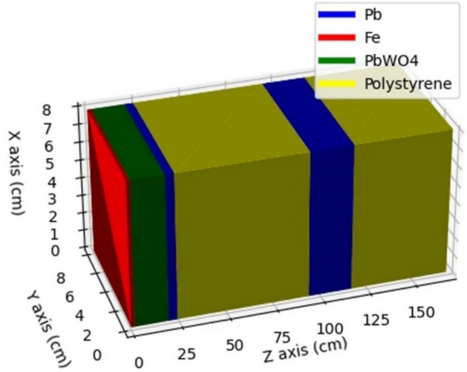


Threshold-based classifier

- Alternative to one-hot encoding restricted to two categories
- Idea: if x < 0 choose PbWO4 else Polystyrene
- Outputs a hybrid calorimeter
  - Lead glass ECAL at the front for short EM showers
  - ‘Sandwich’ Hadronic section at the back for deep pion showers



Starting configuration

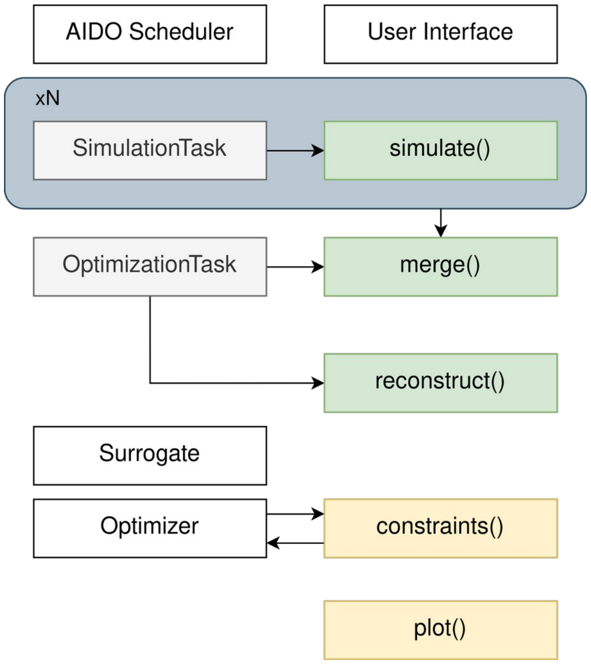


Final configuration

Credit to Nikhil Krishna

Detector Optimization as Python Package

- AIDO software is meant as a framework
- Requires three components:
  - **Simulation**: start geant4 simulation (single-threaded). Save the results in the provided path.
  - **Merge**: build a reconstruction dataset from the simulation outputs
  - **Reconstruction**: compute a loss per event that encapsulates the detector's performance*
- Two optional methods:
  - **Constraints**: add user-defined penalties to the Optimizer loss
  - **Plots**



* source for a lot of trouble

AIDO entry point

- Usage: provide optimizable parameters and UI class
- Parameters are saved as .json files (easy to read in python and C++)

```
if __name__ == "__main__":

    parameters = [
        aido.SimulationParameter("thickness_absorber_0", 5.0, min_value=0.1, sigma=1.5),
        aido.SimulationParameter("thickness_scintillator_0", 5.0, min_value=0.1, sigma=1.5),
        aido.SimulationParameter("material_absorber_0", "G4_Pb", discrete_values=["G4_Pb", "G4_Fe"], cost=[25, 4.166]),
        aido.SimulationParameter(
            "material_scintillator_0",
            "G4_POLYSTYRENE",
            discrete_values=["G4_PbWO4", "G4_POLYSTYRENE"],
            cost=[2500.0, 0.01]
        ),
        aido.SimulationParameter("num_events", 400, optimizable=False),
        aido.SimulationParameter("max_length", 200, optimizable=False),
        aido.SimulationParameter("max_cost", 50_000, optimizable=False),
    ]
    aido.optimize(
        parameters=parameters,
        user_interface=MyUserInterface,
        simulation_tasks=20,
        max_iterations=200,
        threads=20,
        results_dir="results",
        description="""Calorimeter with two layers"""
    )
    os.system("rm *.root")
```

AIDO UserInterface

```python
import aido


class MyUserInterface(aido.AIDOBaseUserInterface):
    """ This class is an example of how to implement the 'AIDOUserInterface' class.
    """

    htc_global_settings = {}

    def simulate(self, parameter_dict_path: str, sim_output_path: str):
        os.system(
            f"singularity exec -B /work,/ceph /ceph/kschmidt/singularity_cache/minicalosim_latest.sif python3 \
            examples/calo_opt/simulation.py {parameter_dict_path} {sim_output_path}"
        )
        return None

    def convert_sim_to_reco(...

    def merge(...

    def reconstruct(self, reco_input_path: str, reco_output_path: str):
        """ Start your reconstruction algorithm from a local container.
        """
        os.system(
            f"singularity exec --nv -B /work,/ceph /ceph/kschmidt/singularity_cache/minicalosim_latest.sif python3 \
            examples/calo_opt/reco_script.py {reco_input_path} {reco_output_path} {self.results_dir}"
        )
        os.system("rm *.pkl")
        return None

    def loss(self, y: torch.Tensor, y_pred: torch.Tensor) -> torch.Tensor:...
```

Summary

- Transposition of detector building task into ML hyper-parameter optimization
- Generalized approach for any geant4 simulation and reconstruction algorithm
- Optimization of discrete parameters using one-hot encoding
- Implementation of additional cost and geometry constraints
- Open-source code compatible for any geant4 simulation software

References

- [1] https://medium.com/hackernoon/gradient-descent-aynk-7cbe95a778da
- [2] https://www.linkedin.com/pulse/diffusion-model-generative-image-synthesis-yogeshwaran-singarasu-jgo4c
- [3] https://www.researchgate.net/figure/interpolation-in-3D-space_fig3_319653273

- AIDO Software: https://gitlab.etp.kit.edu/kschmidt/aido
- Calo-opt (simulation, reconstruction, surrogate, optimizer) https://gitlab.etp.kit.edu/jkiesele/calo-opt
- B2luigi: https://github.com/belle2/b2luigi