LPCC Fastsim

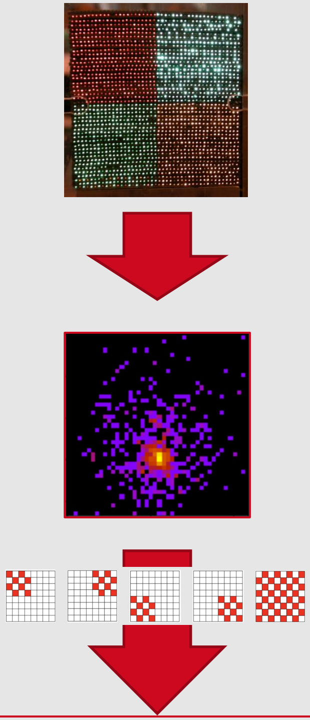
18/11/2021

Jan Dubiński, Kamil Deja, Sandro Wenzel, Roberto Preghenella, Bartosz Świrta, Przemysław Rokita, Tomasz Trzciński

Fast simulation application / accuracy

ALICE Run3:

- in general, no ML fast sim used in production. In process of R&D
- no real outstanding sensitive detector **apart from ZDC** (zero degree calorimeter)
  - currently ZDC not used in all MC productions
- ZDC also appears to be ideal for ML, since it has 2D readout structure (n*m optical fibers collecting photons)
  - a response is a 2D image with pixels encoding photon count
  - the final output are 5 digits calculated from the image (5 channels)
  - Almost no material / other detectors in front of ZDC
  - Approach:
    - generate response directly from primary (no transport at all involved; main one followed up until now)
    - generate response from impinging track (transport involved)
- We target to replace detailed sim with fast sim for ZDC and use in all analysis (if possible)
- Training on actual data not done (possibly not easy; could be interesting idea)



8, 22, 6, 197, 791

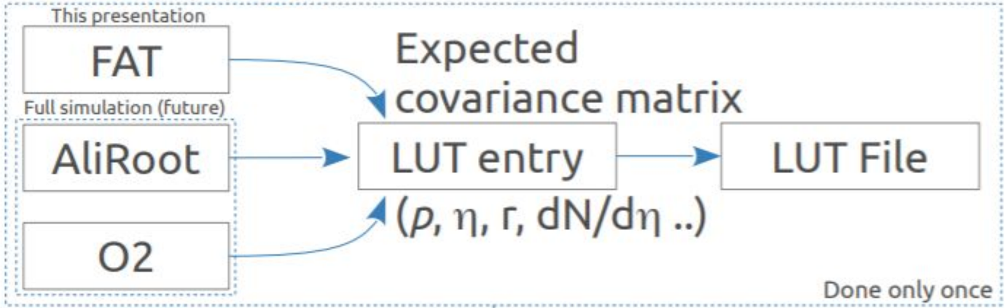
ALICE3: Fast simulation in Delphes spaces

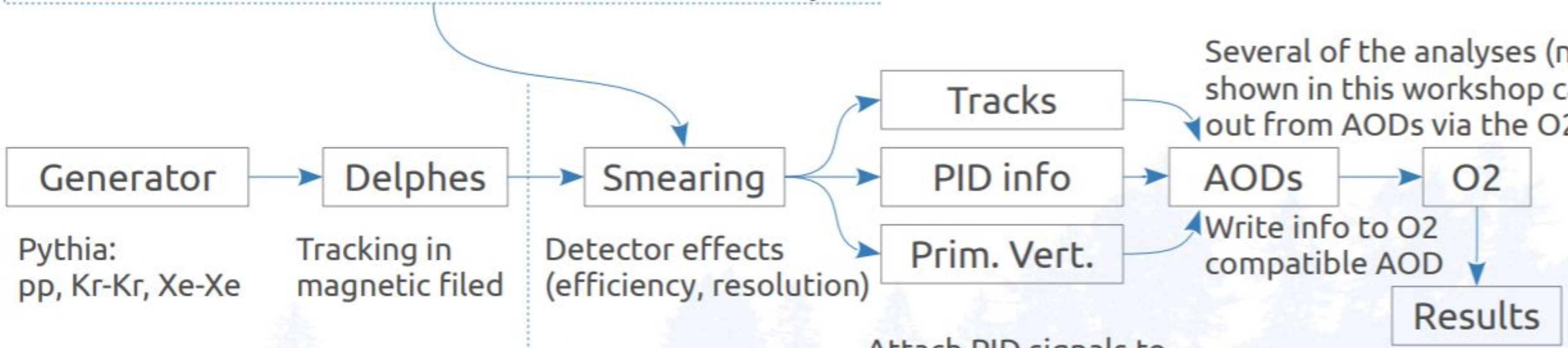
ALICE 3 Upgrade studies

- **new fast-simulation framework developed on purpose (DelphesO2)**
  - based on Delphes
  - and on ALICE-O2 (simulation and reconstruction for Run3/4)
  - plus custom routines for simulation of signal of specific detectors
    - time-of-flight layers
    - RICH detectors
    - EMCal
    - muon identification layer
  - GRID enabled, extremely large data samples (billions of HI collisions)
- **produce analysis objects in the ALICE data format**
  - convert output of fast simulation into AOD data
  - can run same identical analysis code as normal simulation / reconstruction
  - GRID analysis of very large data samples
- **input to DelphesO2**
  - event generator output (pythia8, HepMC, …)
  - LUTs with tracking parameterisation (see later)
  - parameters for fast-simulation of signal of other (i.e. PID) detectors
    - time-of-flight layer(s): time resolution, location, …
    - RICH detector(s): refractive index, 1pe angular resolution, PDE, ...

Practical implementation: DelphesO2

Tracker design





Several of the analyses (not all!) shown in this workshop carried out from AODs via the O2

Pythia: pp, Kr-Kr, Xe-Xe

Tracking in magnetic filed

Detector effects (efficiency, resolution)

Write info to O2 compatible AOD

Attach PID signals to track (TOF, RICH, MuonID, FTOF, FRICH ..)

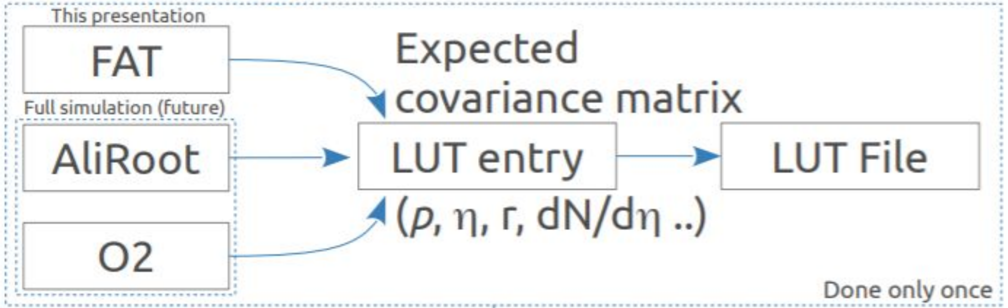
Later analysis: QA, secondary vertex reconstruction, ML ..

FAT: fast analytical tool
LUT: lookup table

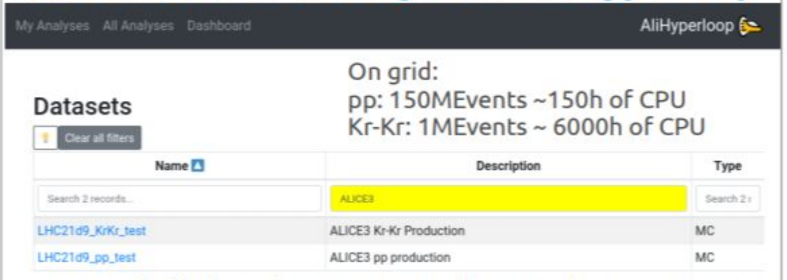
DelphesO2

Practical implementation: DelphesO2

Tracker design



Fast simulation already took the hyperloop

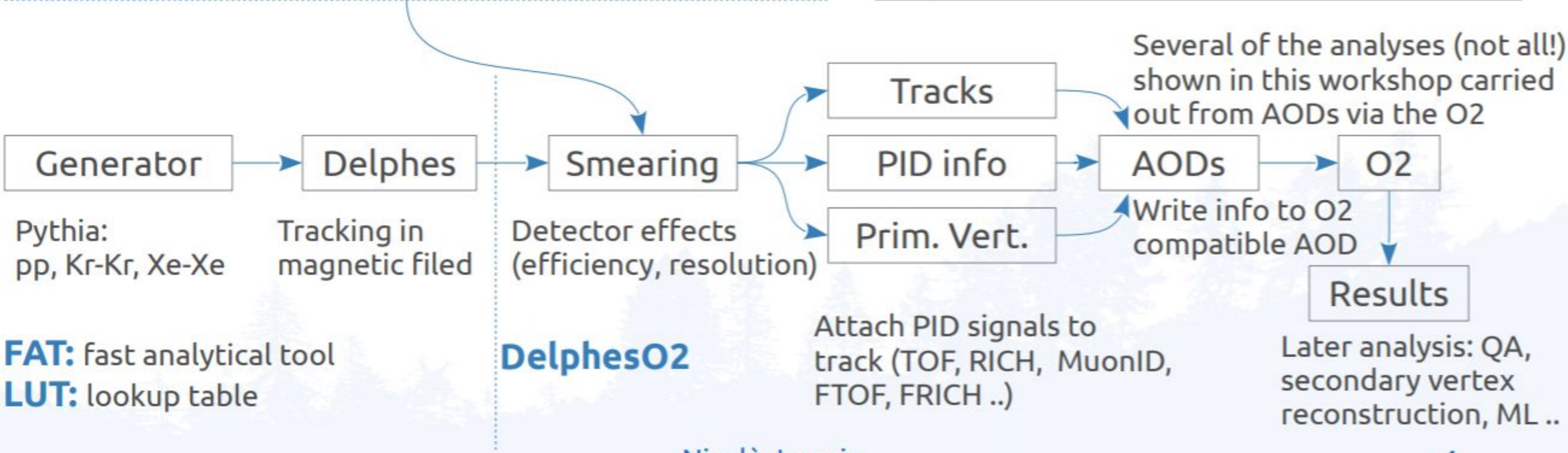


Datasets

Name	Description	Type
Search 2 records...	ALICE3	Search 2 r
LHC21d9_KrKr_test	ALICE3 Kr-Kr Production	MC
LHC21d9_pp_test	ALICE3 pp production	MC

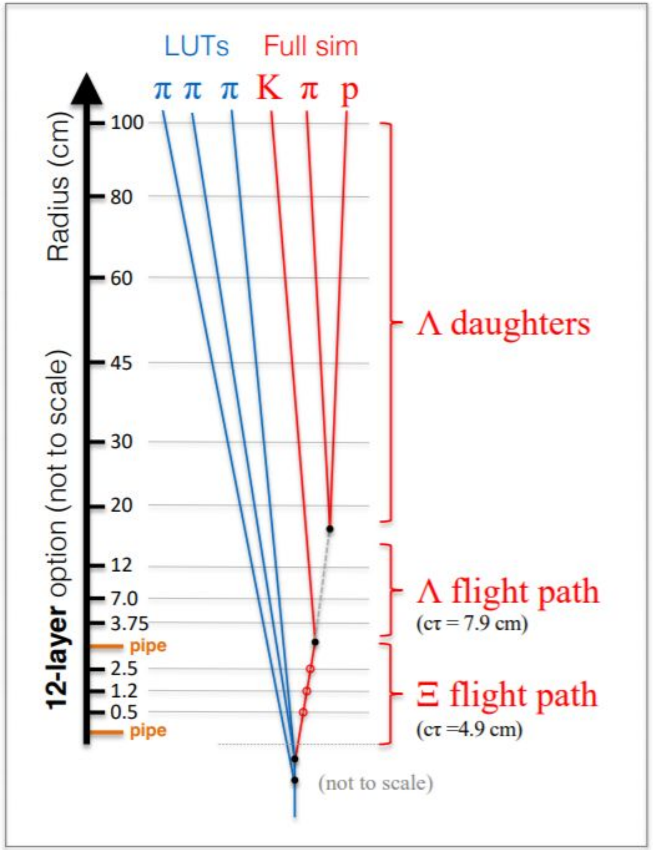
On grid:
pp: 150MEvents ~150h of CPU
Kr-Kr: 1MEvents ~ 6000h of CPU

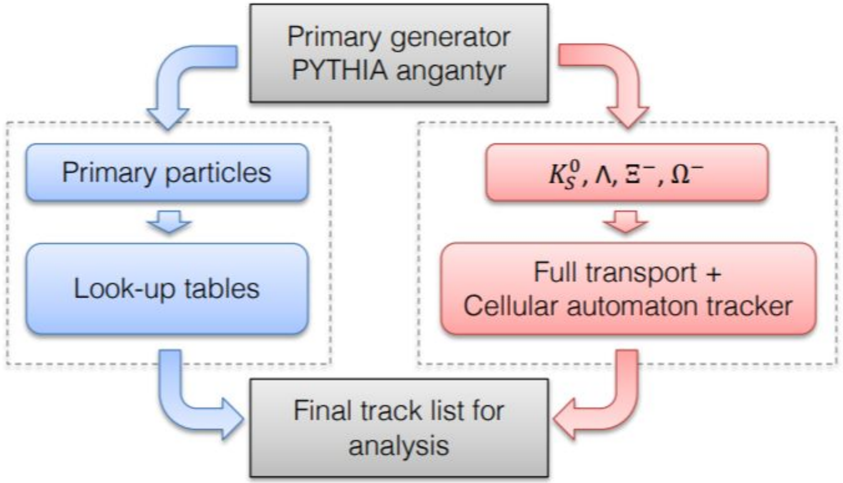
Special thanks to Catalin and Jan Fiete!


Generator
Delphes
Smearing
Tracks
PID info
Prim. Vert.
AODs
O2
Results
Pythia: pp, Kr-Kr, Xe-Xe
Tracking in magnetic filed
Detector effects (efficiency, resolution)
Several of the analyses (not all!) shown in this workshop carried out from AODs via the O2
Write info to O2 compatible AOD
Attach PID signals to track (TOF, RICH, MuonID, FTOF, FRICH ..)
Later analysis: QA, secondary vertex reconstruction, ML ..
DelphesO2

FAT: fast analytical tool
LUT: lookup table

HYBRID technology

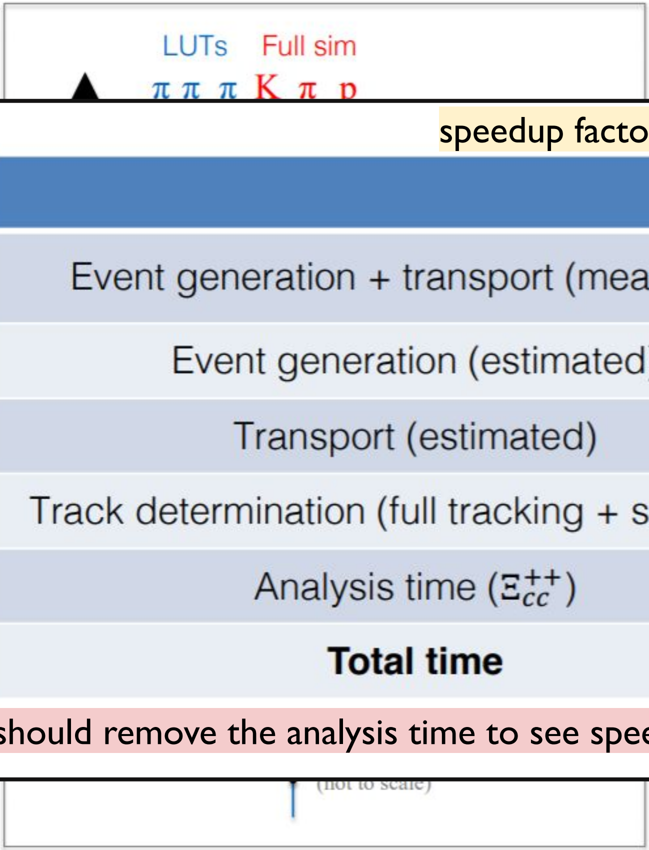




Primary generator PYTHIA angantyr
Primary particles
Look-up tables
$K_S^0, \Lambda, \Xi^-, \Omega^-$
Full transport + Cellular automaton tracker
Final track list for analysis

- LUTs: adequate for primary particles
- Full simulation: adequate for secondaries
- Combination yields high speed: reasonable objective is that time in tracking + smearing is less than analysis and event generation
- Full preservation of weak decay selections vs LUT approach!

HYBRID technology



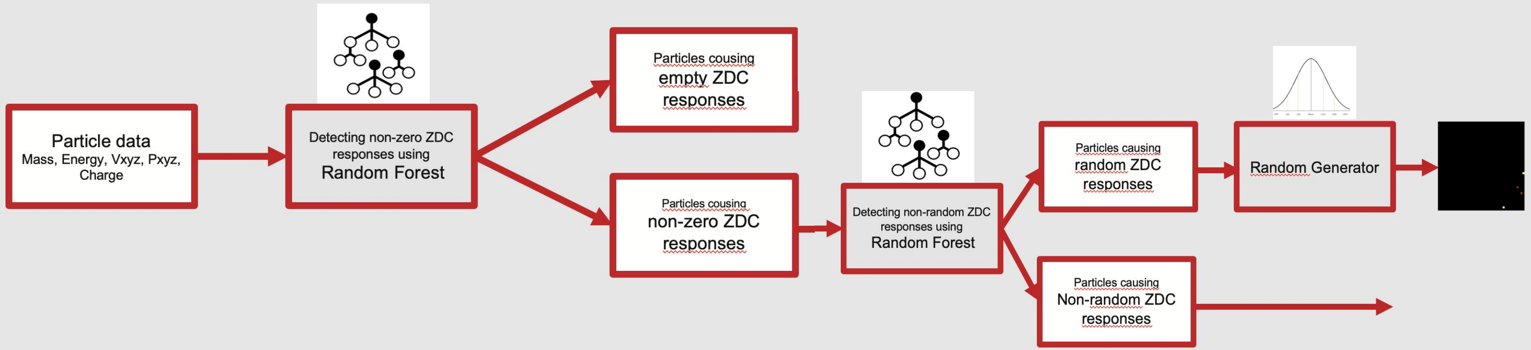
speedup factors (numbers for 100 min.bias Pb-Pb events)

	Full sim	Fraction	Hybrid sim	Fraction
Event generation + transport (measured)	15 minutes	14%	3 minutes	26.5%
Event generation (estimated)	~2.5 minutes	2.3%	~2.5 minutes	22.1%
Transport (estimated)	~12.5 minutes	5.8%	~30 seconds	4.4%
Track determination (full tracking + smearing)	85 minutes	78.7%	20 seconds	2.9%
Analysis time (Ξ_{cc}^{++})	~8 minutes	7.4%	~8 minutes	70.8%
Total time	**108 minutes**	**100%**	**11.3 minutes**	**100%**
should remove the analysis time to see speedup	100 minutes		3.3 minutes	30x faster

generation
- Full preservation of weak decay selections vs LUT approach!

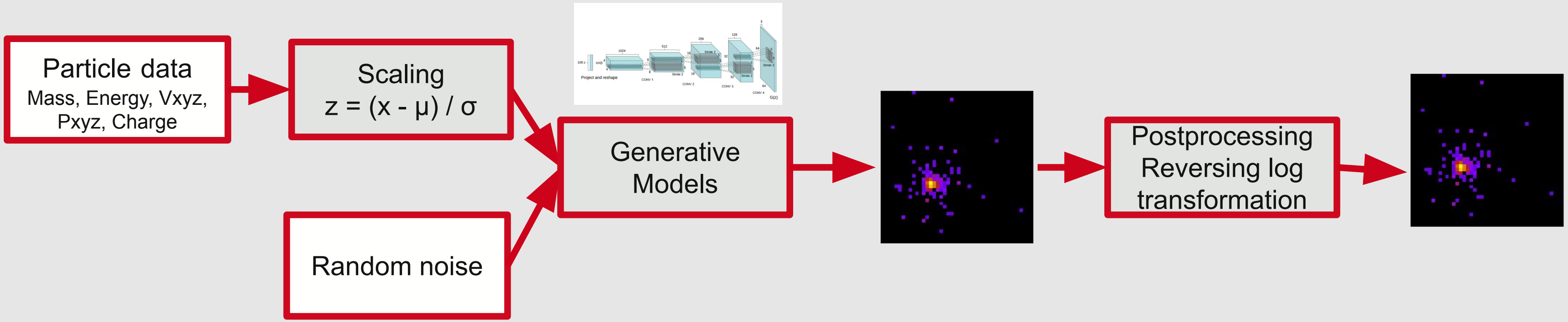
Data preprocessing

- Dataset consists of 10 million particle examples with conditional parameters (Energy, primary vertex position (x, y, z), momenta (x, y, z), mass, charge)
- Instead of inputting particle id (pdg code) we convert them to mass and charge
- We exclude particles producing "empty" or "nearly empty" outputs using trained Random Forest models
- We store preprocessed data as .npz files (zipped archive with numpy arrays)


Particle data
Mass, Energy, Vxyz, Pxyz, Charge
Detecting non-zero ZDC responses using Random Forest
Particles cousing empty ZDC responses
Particles cousing non-zero ZDC responses
Detecting non-random ZDC responses using Random Forest
Particles causing random ZDC responses
Particles causing Non-random ZDC responses
Random Generator

Generating non-random ZDC responses

- The input of the simulation is random noise and conditional parameters (Energy, primary vertex position (x, y, z), momenta (x, y, z), mass, charge)
- The final dataset consists of 295867 examples
- We scale the conditional input with standard scaler
- We transform the ZDC response images with logarithm before using them as real training data



Tuning

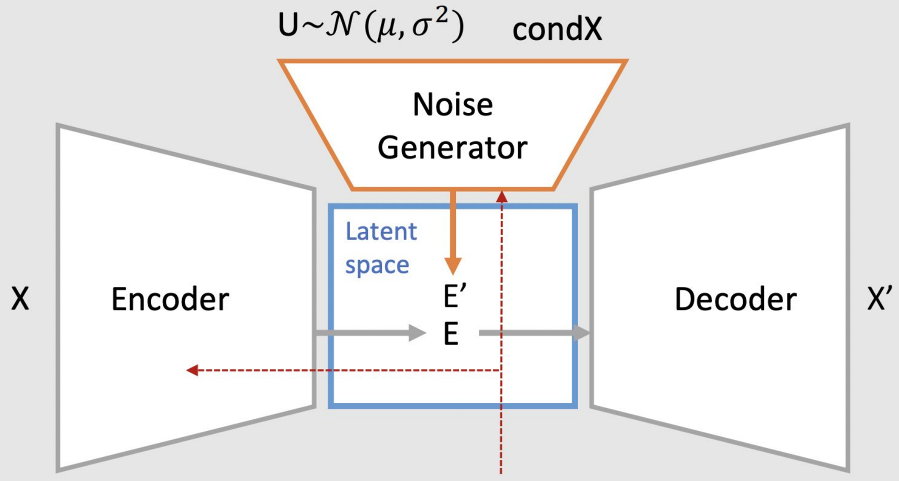
- We reviewed different generative architectures: VAE, GAN and our own idea e2e SAE (see IML presentation and article)
- The final decision was based on the wasserstein distance between channels of original and fast simulation
- Current state of the art architectures:
  - end-to-end Sinkhorn autoencoder (paper and code)
  - deep convolutional GAN with auxiliary regressor and postprocessing (code)
- Manual hyperparameters tuning

end-to-end Sinkhorn autoencoder

- No implicit regularisation of autoencoder's latent space
- Approximation of original data embeddings with deterministic neural network
- Joint optimisation of both neural networks

- Conditional information added to noise generator
- Wasserstein distance between embeddings concatenated with conditional values
- Original data distribution on latent space

- Trained with Adam(lr=0.001)



$$\mathcal{L}_1 = S_{c,\epsilon}([E, condX], [E'(condX), condX])$$

Encoded data

E1	E2	CondX
0.1	0.2	0
0.9	0.8	1
0.15	0.1	0

Wasserstein dist.

Encoded noise

E'1	E'2	CondX
0.16	0.12	0
0.88	0.82	1
0.11	0.18	0

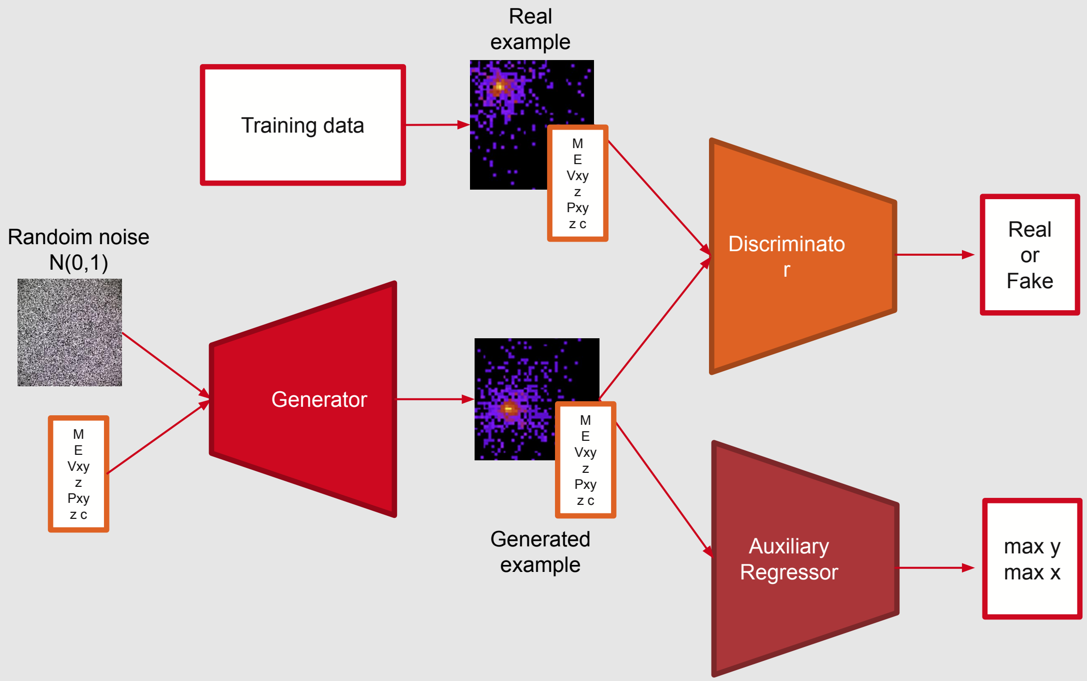
cDC-GAN + auxiliary regressor + postproc

- **An auxiliary regressor** has been added to the pair Discriminator and Generator

- The auxiliary regressor was trained on the task of returning the position coordinates of the maximum number of photons in the input image. After training, the weights of the model were frozen.

- The regressor provides an additional source of loss to the generator by comparing the coordinates of the maximum of the generated examples with the maximum coordinates of corresponding sample in the training set.

- Trained with Adam(lr=0.001)

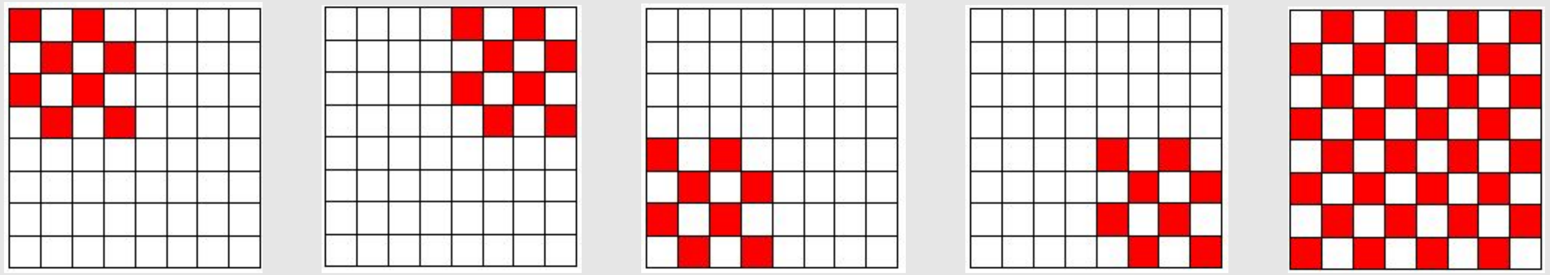


Validation

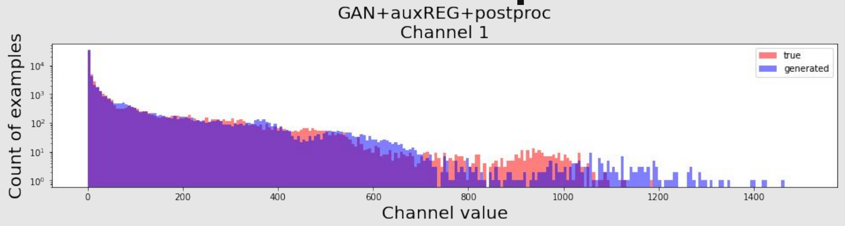
We started with validation on the basis of standard metrics (MSE, differences between placement of central hit for generated and original simulations), but simulations are too random

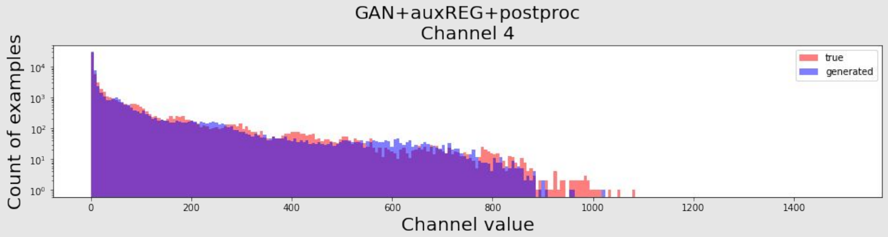
Statistical comparison of output values:

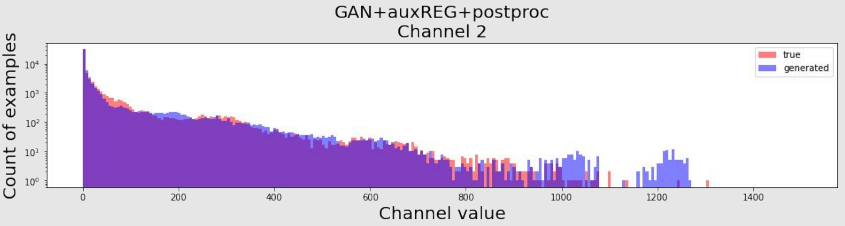
- 5 Channel values are calculated by summing pixels (photons) that are located at the specific fields of a checkerboard grid

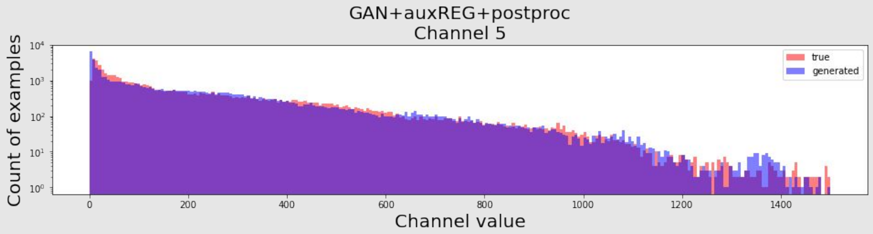


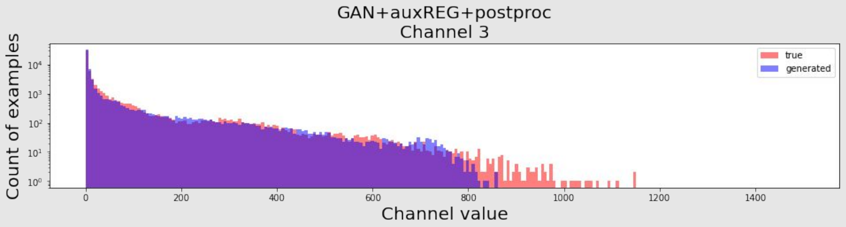
GAN + auxREG + postproc – channel comparision











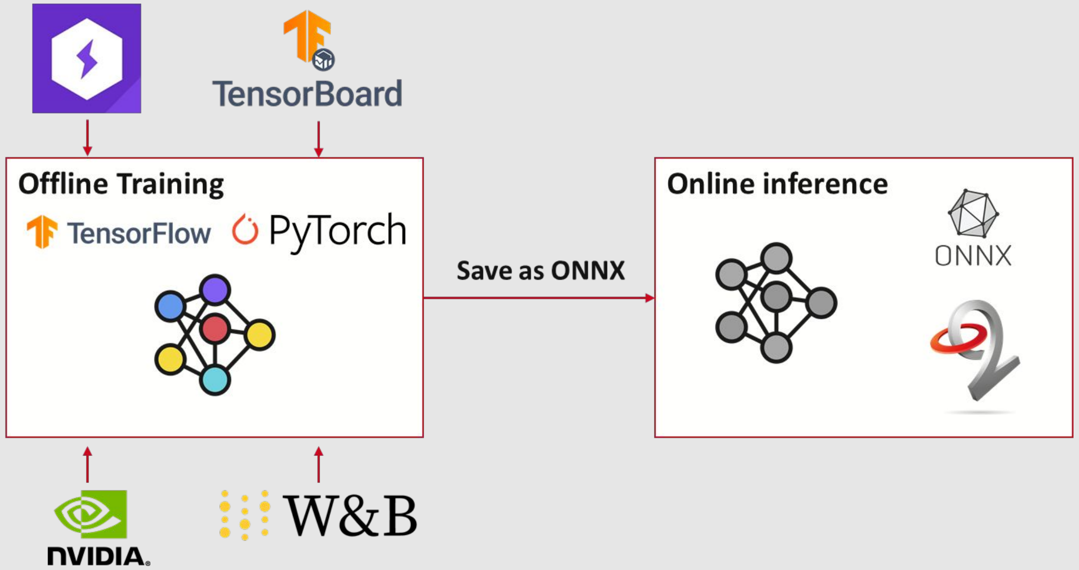
model	WS MEAN	WS CH1	WS CH2	WS CH3	WS CH4	WS CH5
GAN + auxREG + postproc	5.16	2.71	4.63	4.89	6.71	8.59

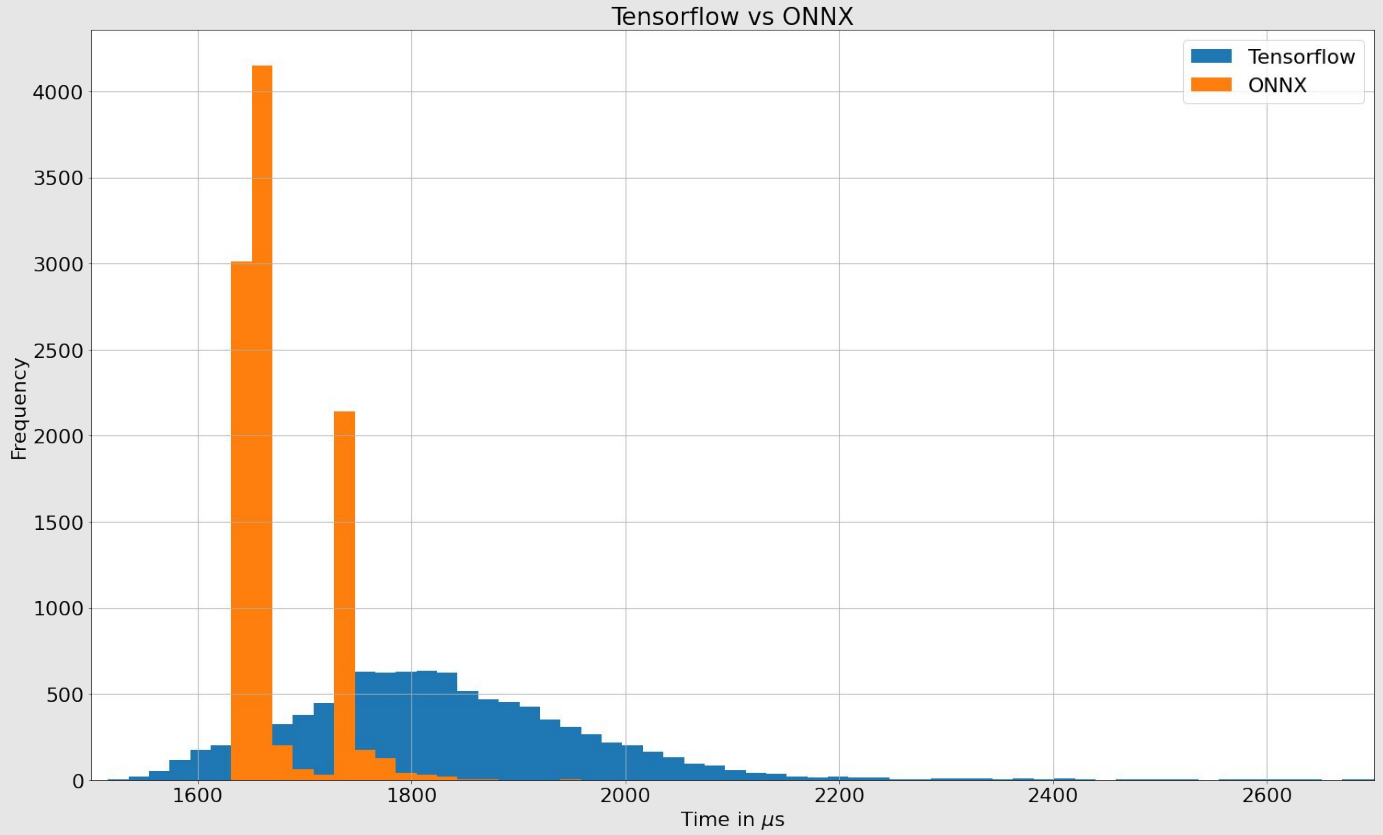
Integration (work in progress)

- We are currently integrating the models into O2
- Models will be stored in ONNX format
- During development and training they are stored in raw TF or PyTorch format
- Generating one particle takes on average 1674 μs on CPU.
- Simulation was not tested on GPU.
- ~ 100MB memory requirements for the whole process marginal compared to rest of data processing

Integration overview

- Offline training where we can use most common ML libraries with addons, and available resources (e.g. GPUs)
- Online inference through ONNX integrated to O2




Tensorflow vs ONNX
Tensorflow
ONNX
Frequency
Time in μs
0
500
1000
1500
2000
2500
3000
3500
4000
1600
1800
2000
2200
2400
2600

Valuable lessons and plans

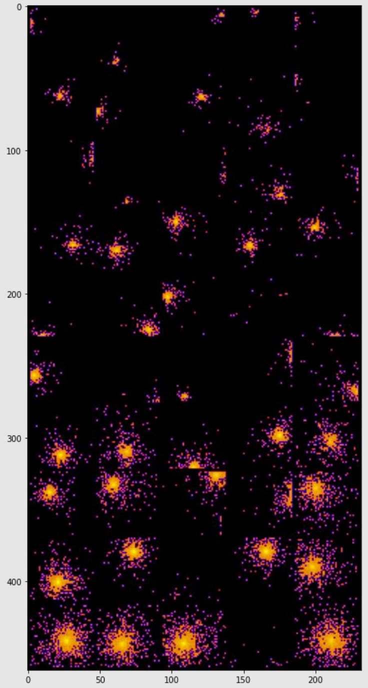
- We can recommend ONNX as an integration tool
- Software we develop is very ZDC specific

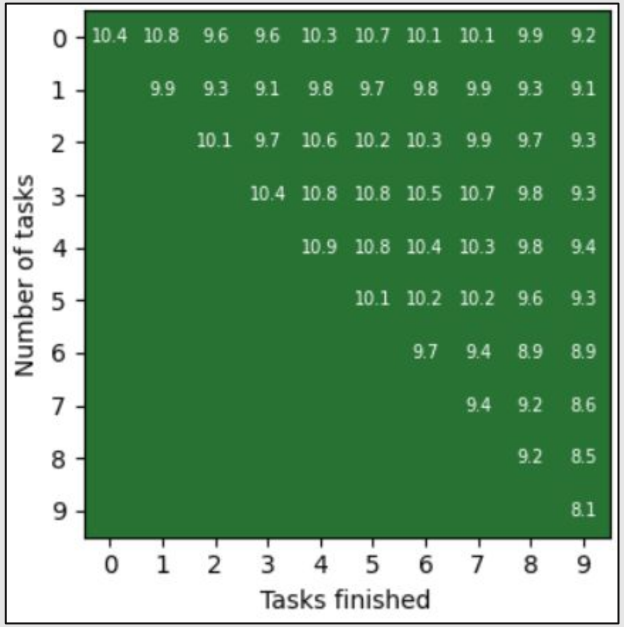
- We might want to think about how we want to retrain model with additional data - we don’t want to do it from scratch every time we have some changes
- Preliminary results with our continual learning method are promising

Continual learning - preliminary results

Training with continuously increasing amount of data

Without the assumption of i.i.d. data





Number of tasks \ Tasks finished	0	1	2	3	4	5	6	7	8	9
0	10.4	10.8	9.6	9.6	10.3	10.7	10.1	10.1	9.9	9.2
1		9.9	9.3	9.1	9.8	9.7	9.8	9.9	9.3	9.1
2			10.1	9.7	10.6	10.2	10.3	9.9	9.7	9.3
3				10.4	10.8	10.8	10.5	10.7	9.8	9.3
4					10.9	10.8	10.4	10.3	9.8	9.4
5						10.1	10.2	10.2	9.6	9.3
6							9.7	9.4	8.9	8.9
7								9.4	9.2	8.6
8									9.2	8.5
9										8.1

Wasserstein distance on all channels with continuously increased amount of data