ML-based improvements for ProtoDUNE HD electronics response fit

Rado Razakamiandra, Karla Tellez-Giron-Flores, Xin Qian

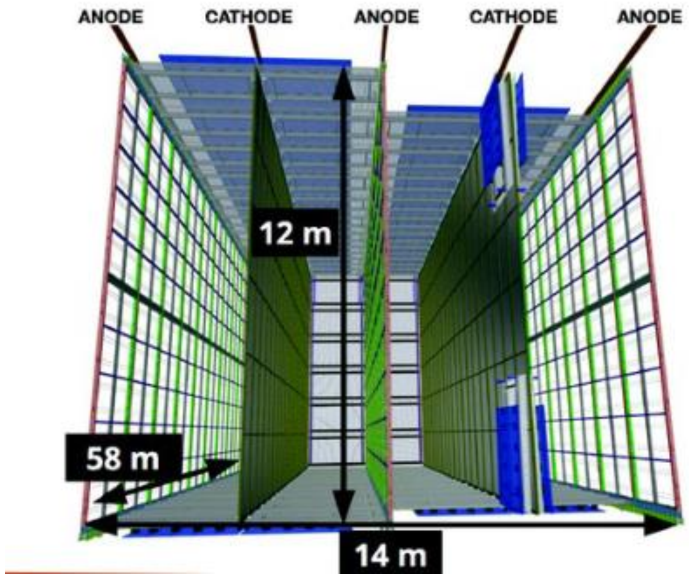
Local ProtoDUNE meeting

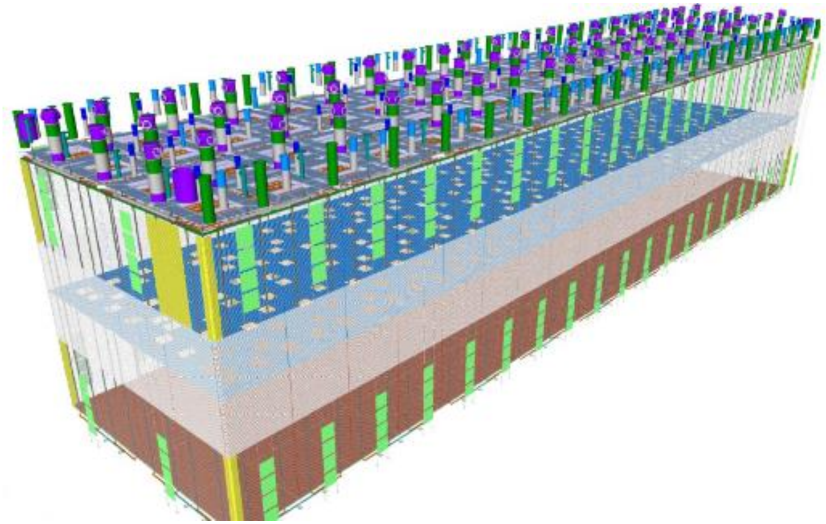
July 9th, 2025

Stony Brook University

Brookhaven National Laboratory

DUNE Far detector Horizontal and Vertical Drifts





- 500 V/cm horizontal drift field
- Wire planes charge read-out at the anode: 2 induction and 1 collection planes.
- 150 Anode Plane Assemblies (APA) in total:
  - 2560 channels / APA

  => **384,000** channels

- 500 V/cm vertical drift field
- PCB charge read-out: 2 induction and 1 collection planes.
- 160 Charge Readout Planes (CRPs) in total:
  - 3072 channels / CRP

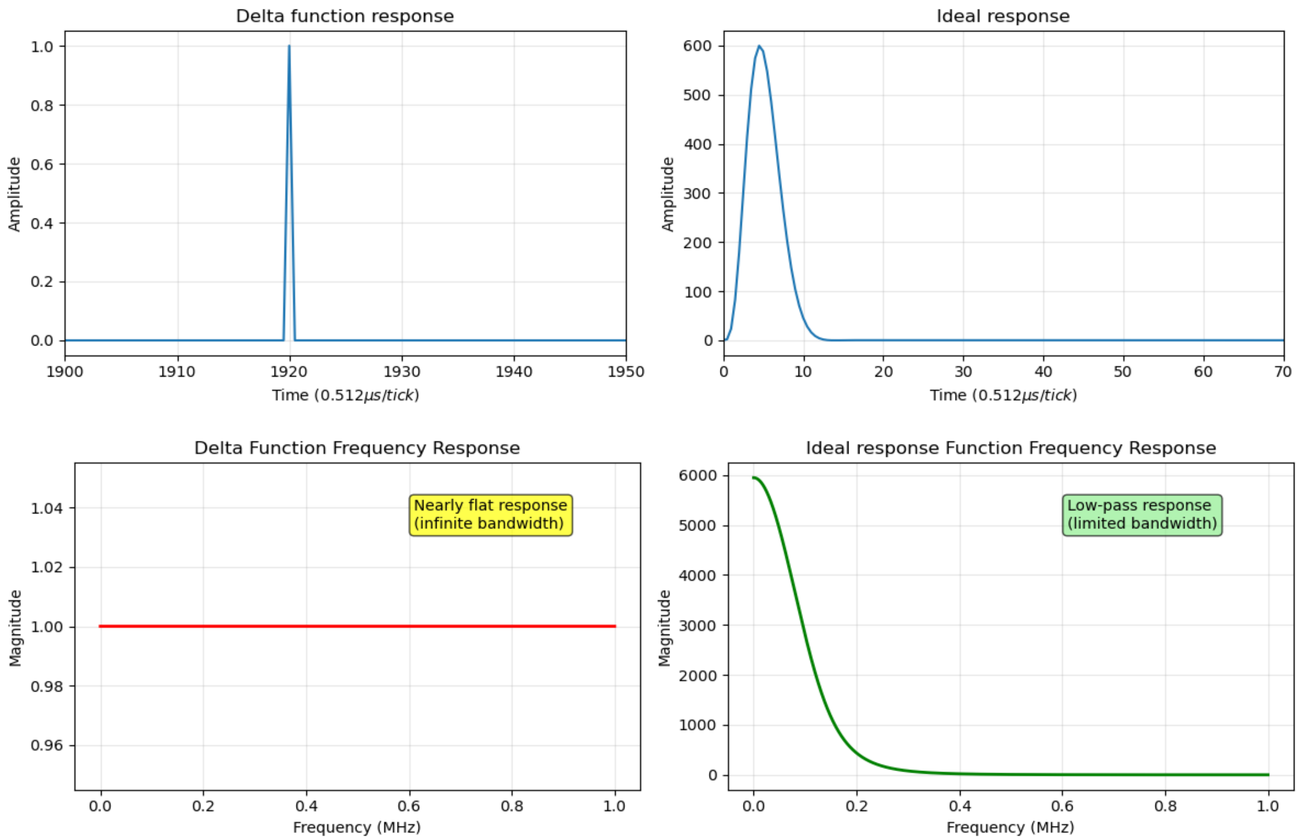
  => **491,520** channels

- A dedicated quality control is needed to minimize the dead channels (<1% is the requirement)and have an electronic system reliable during the lifetime of the experiment.

- **Challenge is the vast number of channels .**

Electronics Response

- Electronic response :
  - inverse Laplace transformation of the transfer function of the pre-amplifier.
  - Convoluted with input induced current for optimal signal-to-noise ratio.
  - Should match the original signal's bandwidth so that it doesn't affect the signal but only limit noise.



- Needed to extract the original signal S through signal processing via deconvolution . where M is the measured signal and R is the electronic response.

$$S(\omega) = \frac{M(\omega)}{R(\omega)}.$$

- **Calibration allows us to get R using an external signal generator.**

Ideal response function

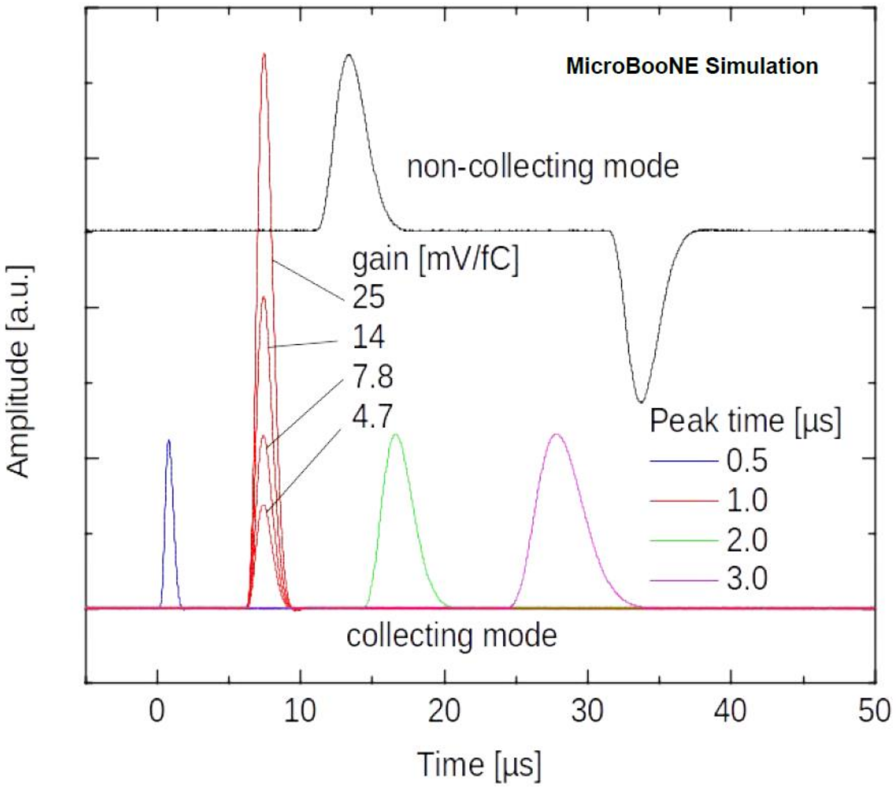
Inverse Laplace Transformation of the transfer function

$$T(s) = \frac{A_0 \cdot C_A}{(p_0 + s) \cdot (p_{i1}^2 + (p_{r1} + s)^2) \cdot (p_{i2}^2 + (p_{r2} + s)^2)}, \tag{3.1}$$

with s being a complex frequency variable. The parameters in equation 3.1 are obtained from a detailed simulation of the network design and are determined to be:

$$\begin{aligned} p_{r1} &= \frac{1.417}{t_p \cdot C_T}, \quad p_{r2} = \frac{1.204}{t_p \cdot C_T}, \\ p_{i1} &= \frac{0.598}{t_p \cdot C_T}, \quad p_{i2} = \frac{1.299}{t_p \cdot C_T}, \\ p_0 &= \frac{1.477}{t_p \cdot C_T}, \quad C_A = \frac{2.7433}{(t_p \cdot C_T)^4}, \\ C_T &= \frac{1}{1.996}; \end{aligned} \tag{3.2}$$

where A_0 is the gain parameter and t_p is the peaking time constant. $T(s)$ has units of $\frac{\mathrm{V}}{\mathrm{C}}\,(\mathrm{Hz})^{-1}$.



[1705.07341] Noise Characterization and Filtering in the MicroBooNE Liquid Argon TPC (arxiv.org)

Second motivation of the calibration

- Pole-zero cancellation:
  - Pole: instability point of the transfer function.
  - Zero: point where the transfer function is zero.
  - Pole-zero cancellation: placing zeros at unstable poles to cancel them out.
    - **Perfect cancellation is difficult in practice especially at cryogenic temperature.**

- Hardware:

Transfer function of the ideal response:

$$T(s) = \frac{A_0 \cdot C_A}{(p_0 + s) \cdot (p_{i1}^2 + (p_{r1} + s)^2) \cdot (p_{i2}^2 + (p_{r2} + s)^2)},$$

Two RC filters are used to remove the baseline from the pre-amplifier and intermediate amplifier : cancelling the unstable poles of the ideal response function.

Time-domain impulse of response of one RC filter: $R_{RC}(t) = \delta(t) - (e^{-t/t_0}/t_0)$

Transfer function of the pre-amplifier with the two RC filters: $T'(s) = T(s) \times T_{RC}(s) \times T_{RC}(s)$

- A perfect cancellation would give the desired response.

$$T(s) = \frac{A}{(p_0 + s)(p_{i_1}^2 + (p_{r_1} + s)^2)(p_{i_2}^2 + (p_{r_2} + s)^2)}$$

- Effect of the RC filters: long tail in the impulse response
  - imperfect pole-zero cancellation.

$$T(s) = \frac{A}{(p_0 + s)(p_{i_1}^2 + (p_{r_1} + s)^2)(p_{i_2}^2 + (p_{r_2} + s)^2)} \cdot \frac{(k_3 + s)(k_5 + s)}{(k_4 + s)(k_6 + s)}$$

  - Needs calibration to determine the non-uniform response and replace it with the ideal response

$$M_i^{corr}(\omega) = M_i(\omega) . \frac{R_{ideal}(\omega)}{R_i(\omega)}$$

    - Ensure the 2D deconvolution (time + wire) can be performed.

Motivation of having a robust fit process

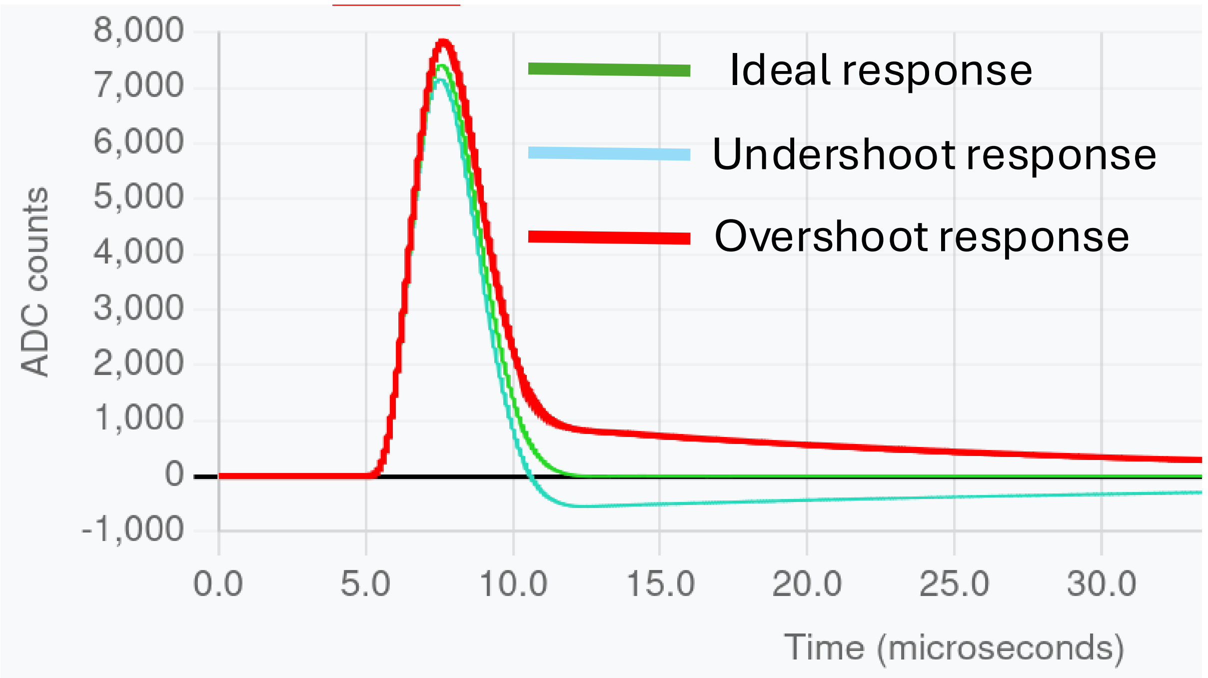
Saving all 310-ticks (used in MicroBooNE) waveforms for all channels of DUNE might consume too much memory

- We can also directly use the analytical form (310 --> 7) and saving the parameters of the fit can help solving that issue.

<u>Response function</u>: inverse Laplace transformation of the modified transfer function

- 6-parameters analytical form: gain, shaping time, k3, k4, k5, k6
- k3 = k4 and k5=k6 gives the ideal waveform.
- Other combinations give waveforms deviated from the ideal.

- Challenge is again 1-million channels, and fit non-linear functions
  - => need a more robust fit process to get a reliable fit result.



Pre-classification and classification

- Provide guidance for the initial fit parameters --> pre-classification

Existing fit procedure:

- If the class of the waveform is known in advance, a set of known values are assigned to the initial values of the fit parameters helping the fit to converge.
- If not, the convergence of the fit is not guaranteed, giving unreliable results.

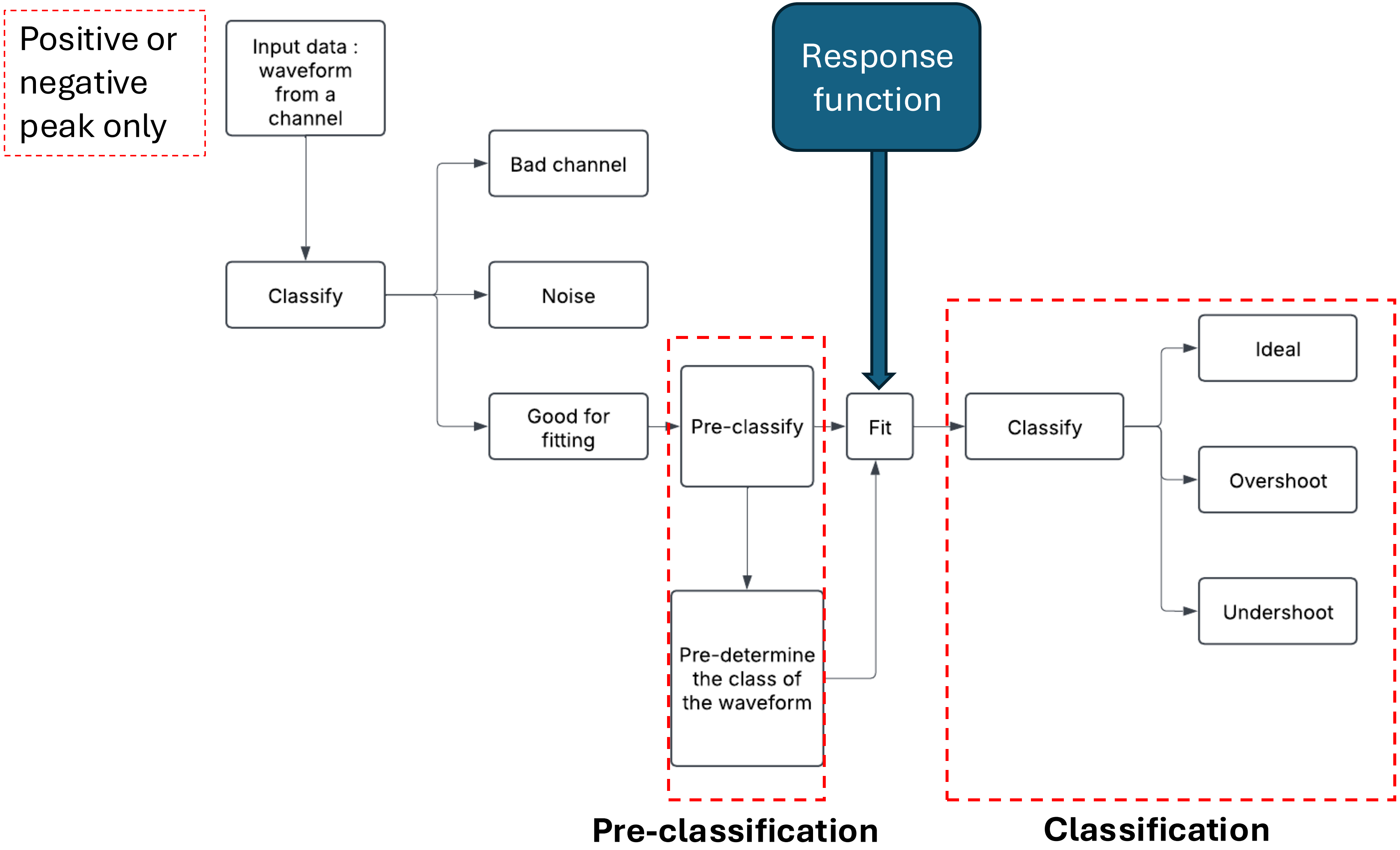
Need a pre-classification stage:

- Because of the large number of channel responses to be fitted in each run, it is crucial to automatize their classifications.

Although the pre-classification might be enough to have a converging fit, we also **need ways to validate** whether that pre-classifier did its job correctly. That validation can be:

- Visualization using a web interface
- (Re-)**classification** of the metrics characterizing the tails of the waveforms.
- Chi2 and Kolmogorov-Smirnov tests
- **OR** all the previous points at the same time

Plan



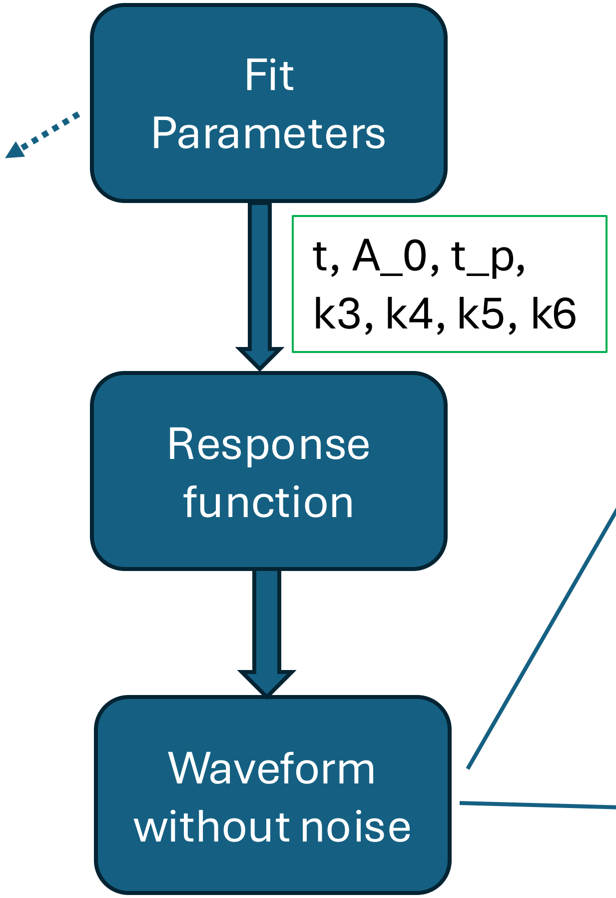
- Pre-classification:
  - taking the waveforms as input, determine its class.
- Classification:
  - taking the fit parameters as input, determine the corresponding class.

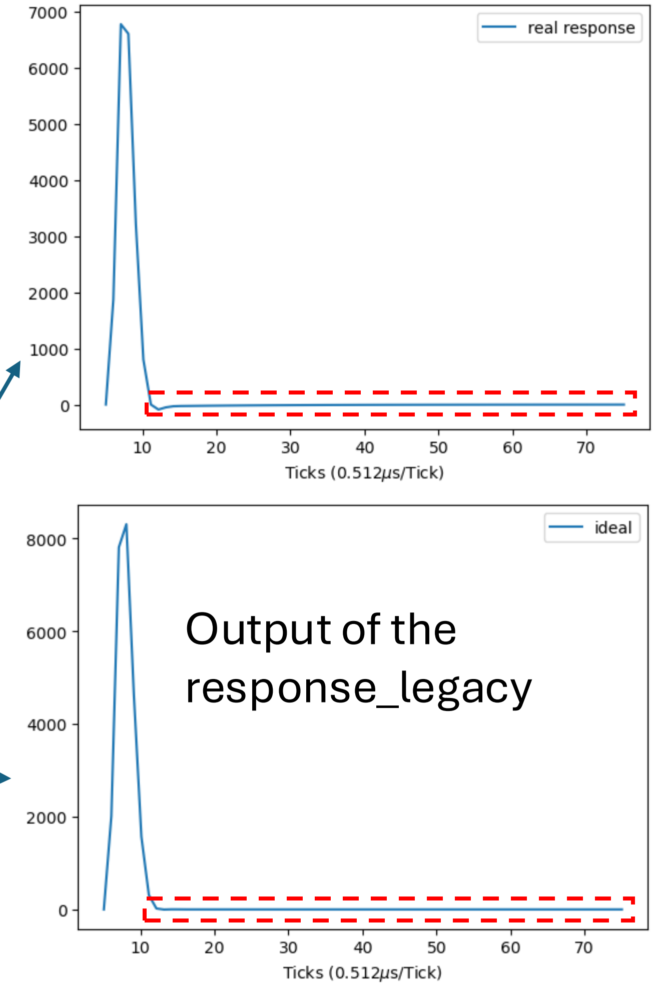
Choice of model: we choose to use a Boosted Decision Tree because of
- the simplicity of the input data.
- Its classification power observed in other applications ([1] and [2]).

[1] Particle Identification Using Boosted Decision Trees in the Semi-Digital Hadronic Calorimeter Prototype, https://doi.org/10.48550/arXiv.2004.02972
[2] Gradient Boosted Decision Tree for Particle Identification Problem at MPD, 10.1134/S1547477125700256

Dataset labelling

```
['#Ch.#',
 't',
 'E(t)',
 'A_0',
 'E(A_0)',
 't_p',
 'E(t_p)',
 'k3',
 'E(k3)',
 'k4',
 'E(k4)',
 'k5',
 'E(k5)',
 'k6',
 'E(k6)',
 'Chi2/DOF',
```

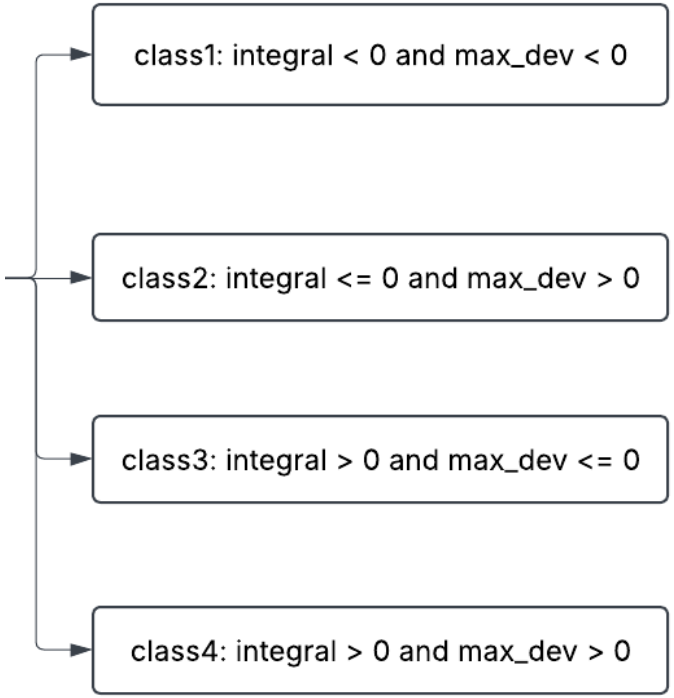




Metrics characterizing the tails:

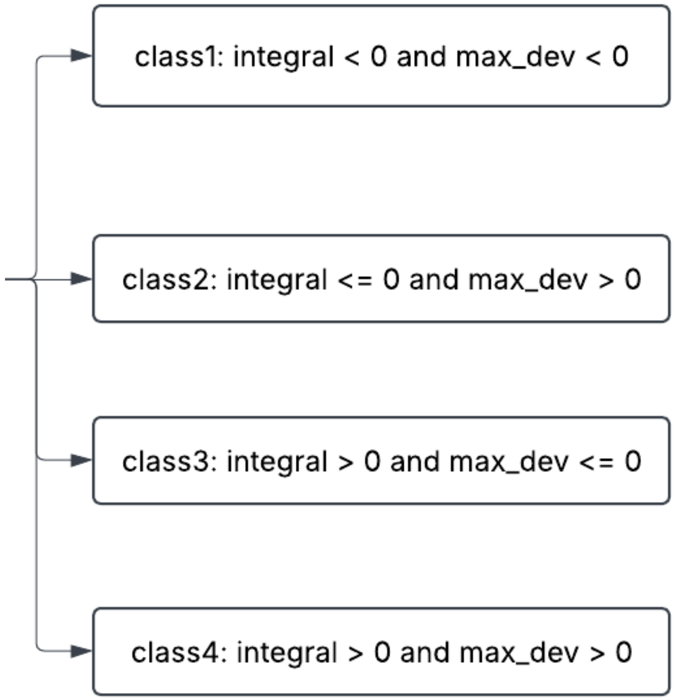
- the integral of the real response's tail.
- Maximum (or minimum if negative) deviation between the tails of the real and the ideal responses.

Labelling the dataset:



Class matching to words we understand

Labelling the dataset:



Undershoot only

Undershoot with small overshoot

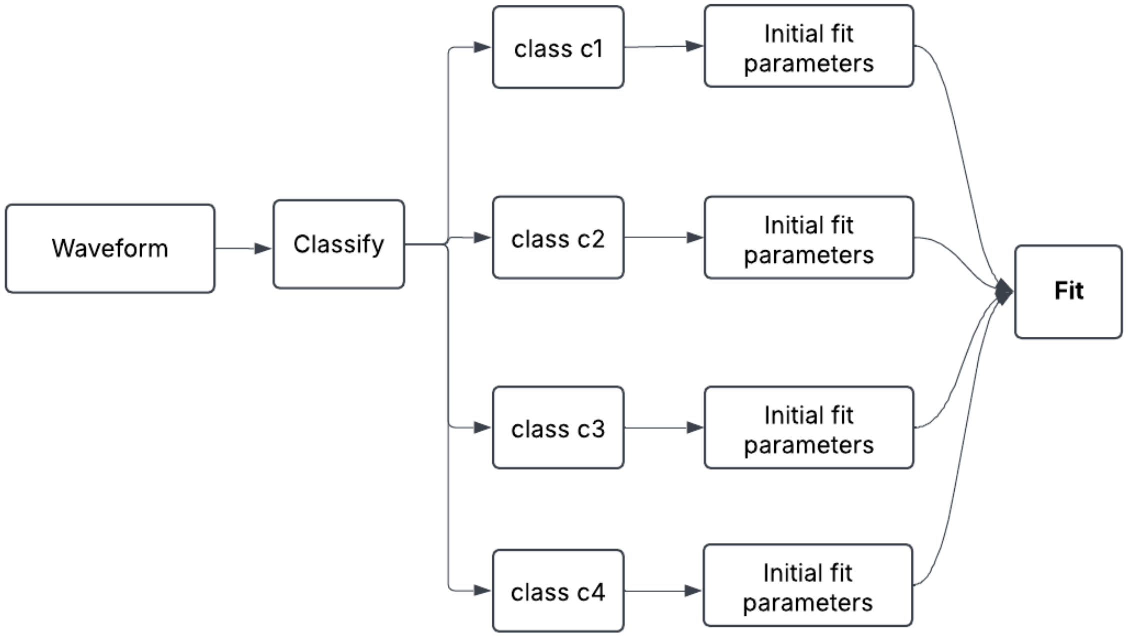
Overshoot with small undershoot

Overshoot only

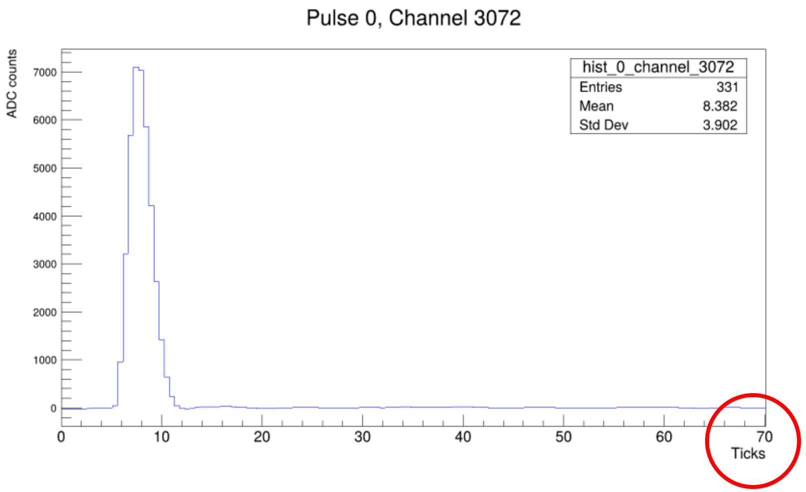
- This information is useful especially in the web interface for visualization.
- The classes c1, c2, c3, c4 are used during the pre-classification and classification. These will be matched with their meaning later in the slide.

Pre-classification

- Model: Boosted Decision Tree implemented with XGBoost
- Input: waveforms
  - The waveforms are generated from the simulated fit parameters.
- Output: classes c1, c2, c3, and c4
- Testing :
  - The waveforms generated from the simulated fit parameters, not used during training.
  - TH1D waveforms from ROOT file : data



70-ticks waveforms used : not really a choice but the maximum ticks in the waveform from the data I received when starting the work.



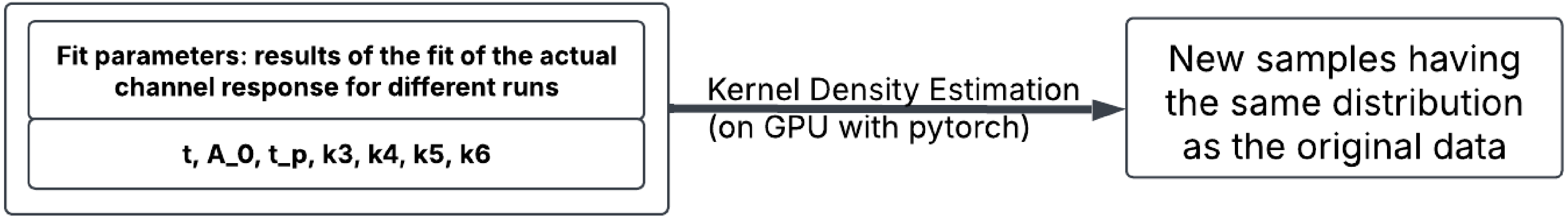
Generation of the simulated fit parameters: used Kernel Density Estimation.

- Can generate samples from a multivariate distribution (distributions of the 7 parameters).
- First method used to generate the new samples and got a good agreement between simulation and data.

Dataset generation using Kernel Density Estimation

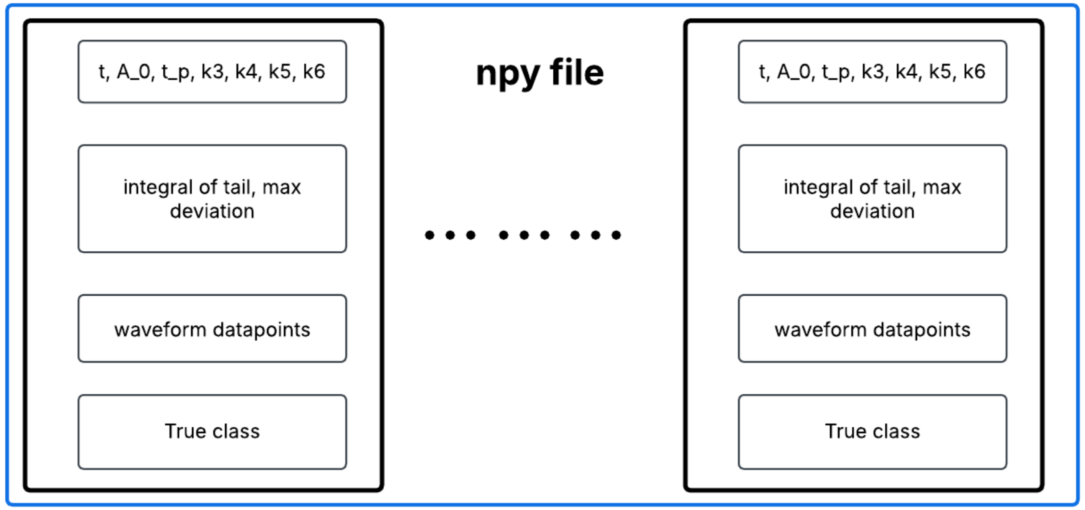
- Motivations:
  - Use generated samples to train the Boosted Decision Trees.
  - Use the actual results of the fit for testing (data).
- Factors required for the generated samples:
  - Should have similar distributions as the original data.
  - Should respect the correlation between parameters.

- How KDE works ?
  - Initialize a model by setting the bandwidth and the datapoints to estimate from.
  - For each set of variables, generate a normal distribution centered at each variable's value with standard deviation equals to the bandwidth.
    - => the smaller the bandwidth, the more accurate the sampling is, but more difficult (takes more time) to generate new samples.
- The generated samples here correspond to the parameters of the fit: t, A_0, t_p, k3, k4, k5,k6



Dataset used for the pre-classification

- The waveforms corresponding to the fit parameters are generated using the response function.
- The fit parameters, metrics (integral of tail and max deviations), and true class are saved in a .npy file.
- Only the **waveform** and the **class** are used for the actual training of the pre-classifier.
    - How many bins? (70 ticks)



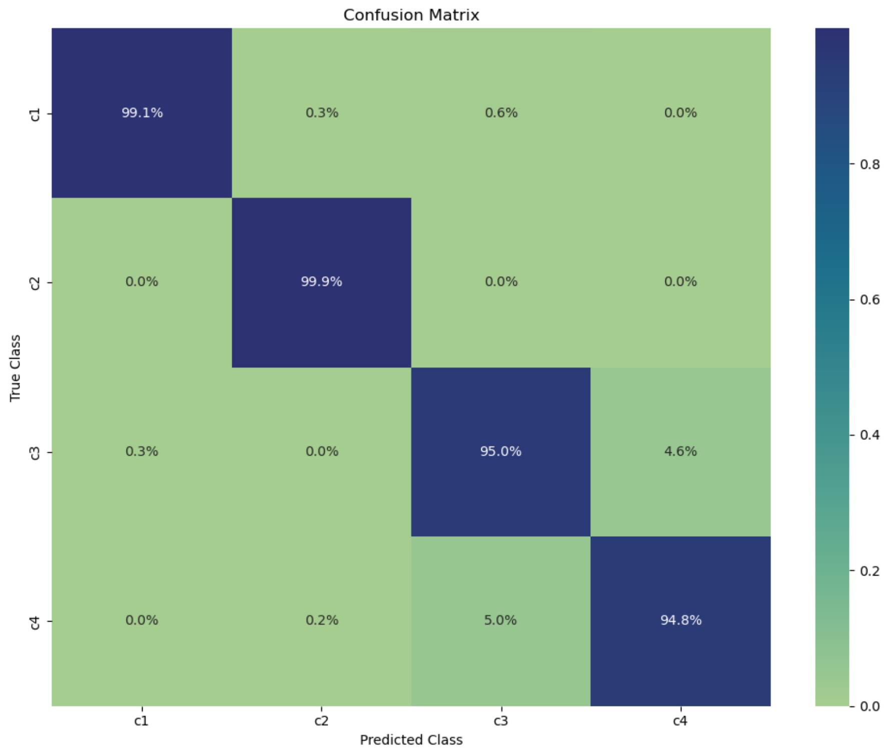
Pre-classification: Training

Input: waveform datapoints
Output: class

Splitting of the dataset into:
- **Training + validation: 640000**
- **Testing: 160000, not used during training**

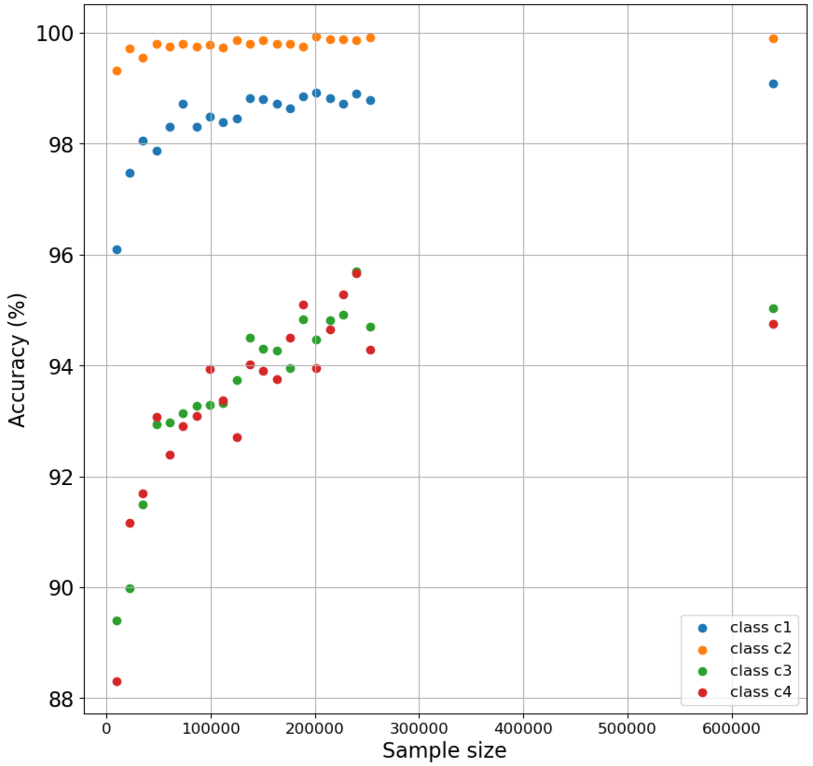
- The prediction accuracy of each class is > 94%.
- Overall classification accuracy on the testing samples: ~97%.
- A test on the real data from ROOT file is done in the next slide.

Confusion matrix of the testing samples:

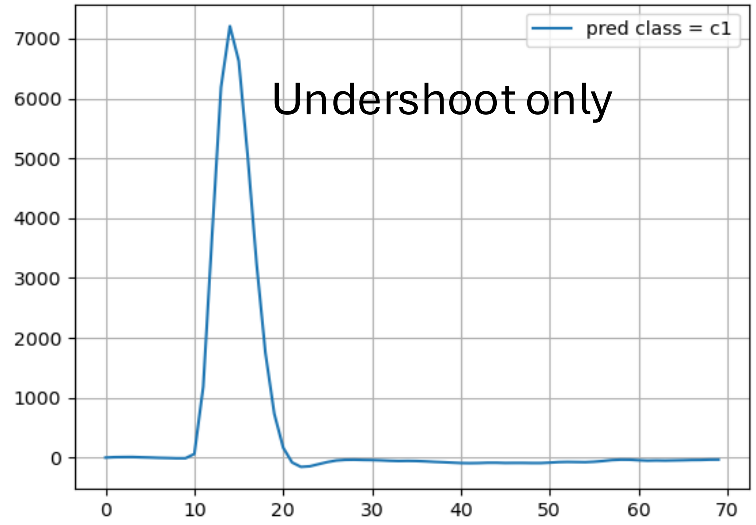


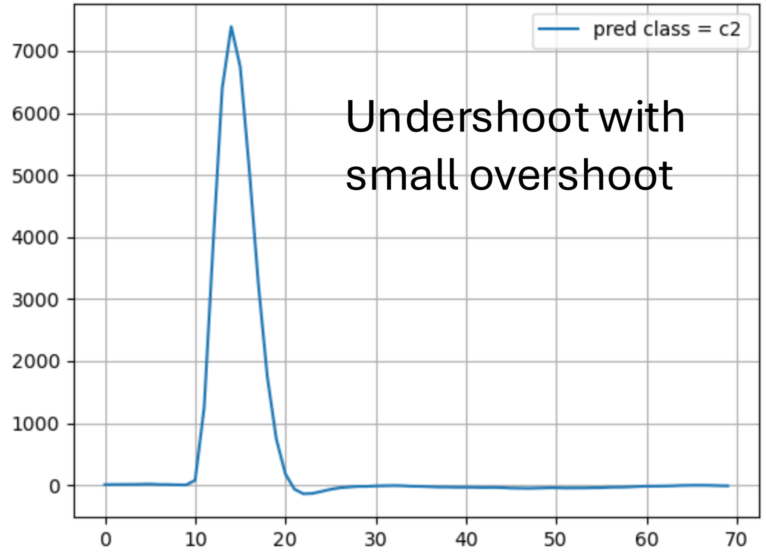
- The confusion between c1 (c3) and c2 (c4) is likely due the similarity between these classes.
- The consequence of these confusions on the fitting is to be understood by running the fitter.

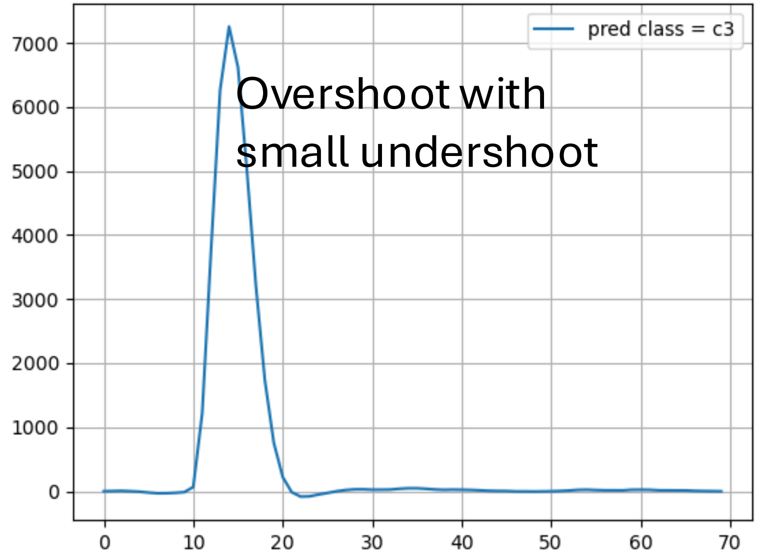
Variation of the accuracies as a function of the training sample size

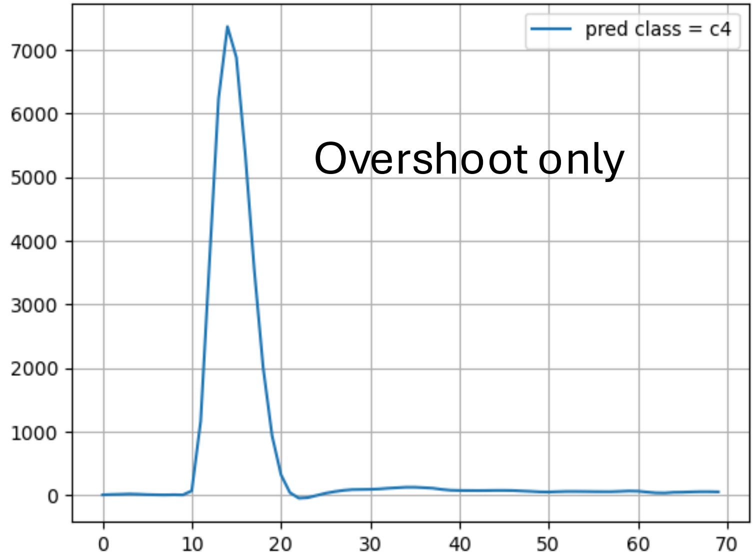


Pre-classification: test on waveforms from ROOT file





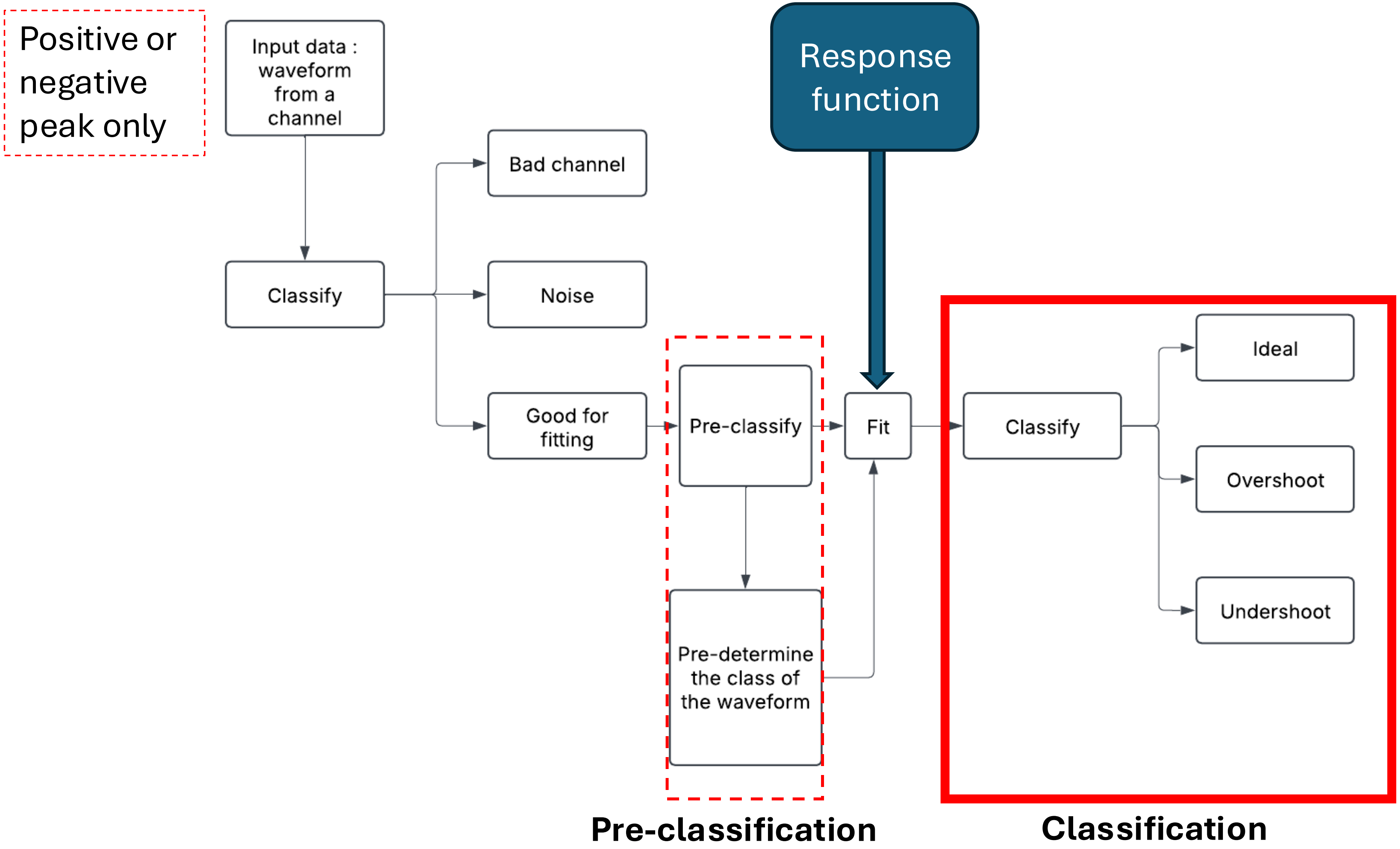




- Input: TH1D from ROOT file.
- From visual inspection, the classes of the waveforms are predicted correctly.

- Confusion seen in previous result:
  - c1 and c2 are similar.
  - c3 and c4 are similar.

Plan



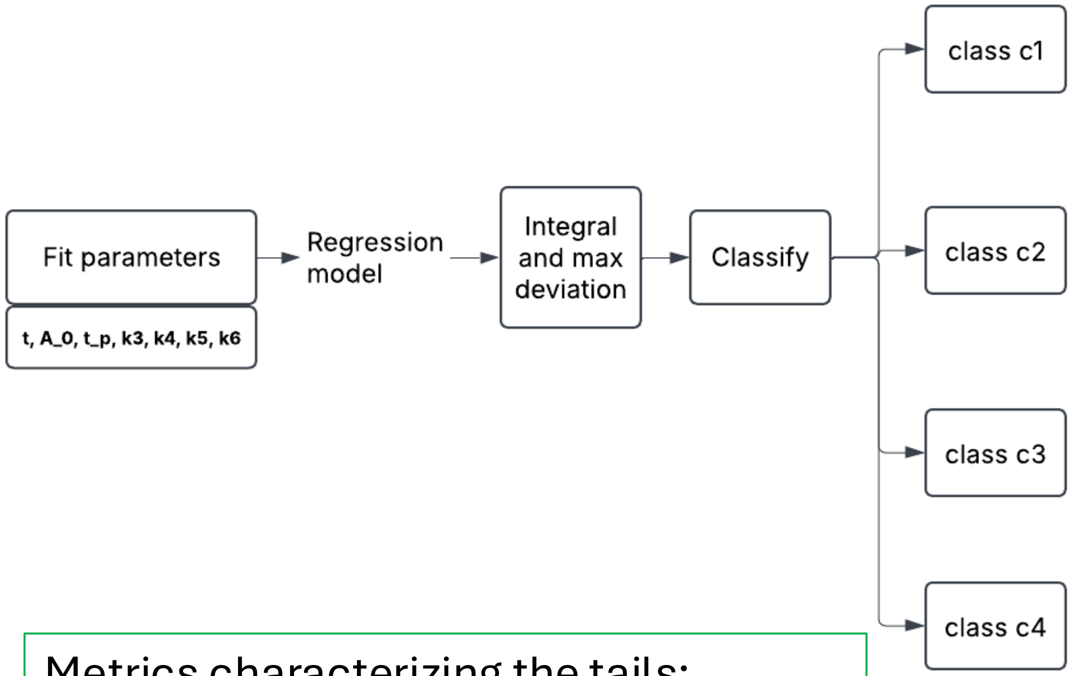
- Pre-classification:
  - taking the waveforms as input, determine its class.
- **Classification:**
  - **taking the fit parameters as input, determine the corresponding class.**

Choice of model: we choose to use a Boosted Decision Tree because of
- the simplicity of the input data.
- Its classification power observed in other applications ([1] and [2]).

[1] Particle Identification Using Boosted Decision Trees in the Semi-Digital Hadronic Calorimeter Prototype, https://doi.org/10.48550/arXiv.2004.02972
[2] Gradient Boosted Decision Tree for Particle Identification Problem at MPD, 10.1134/S1547477125700256

Classification

- Model: Boosted Decision Tree implemented with XGBoost
- Input: waveforms
  - 7 fit parameters
- Output1 : integral and max deviations
- Output2: classes c1, c2, c3, c4



Generation of the simulated fit parameters: used Kernel Density Estimation.

- Can generate samples from a multivariate distribution (distributions of the 7 parameters).

Metrics characterizing the tails:

- the integral of the real response's tail.
- Maximum (or minimum if negative) deviation between the tails of the real and the ideal responses.

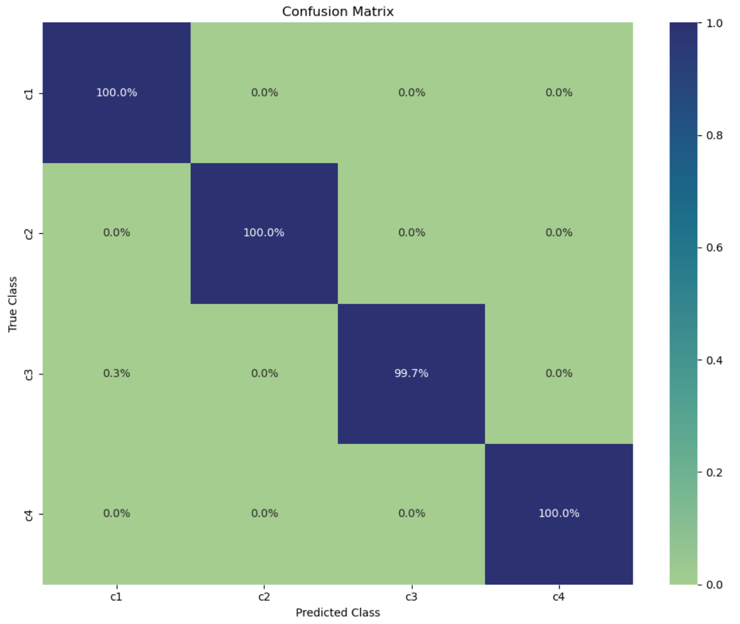
Classification of the labelled fit results (data)

Large imbalance on the dataset

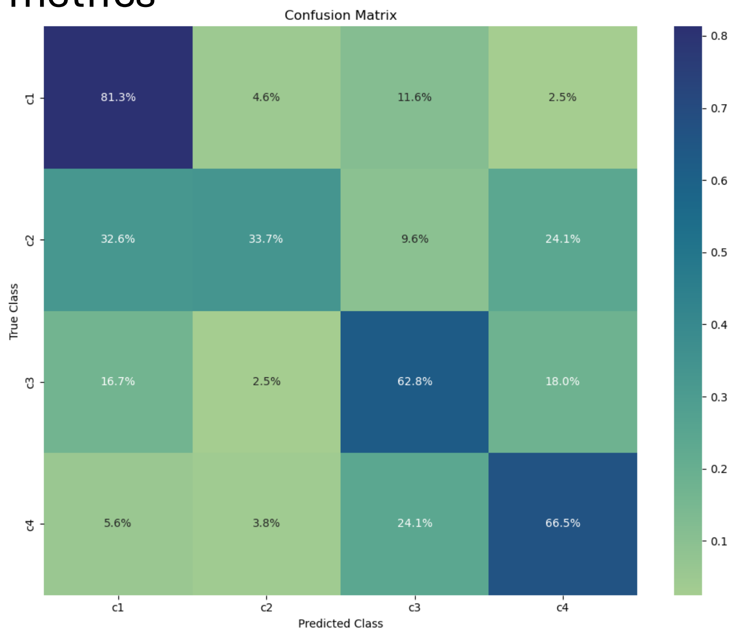
```
train c1: 114665
train c2: 4312
train c3 : 49487
train c4: 60910
```

```
test c1 : 28666
test c2: 1078
test c3: 12371
test c4: 15227
```

Classification of the calculated metrics



Classification of the predicted metrics



Possible reason of the low accuracy:

- Hyperparameters not tune ? The hyperparameters in those models are tuned : using grid search to determine the initial values, then using random search to determine the values closest to the best values.
- Similarities between c1 and c2, and c3 and c4.
- The imbalance on the dataset : the models learn more about the classes with large numbers.

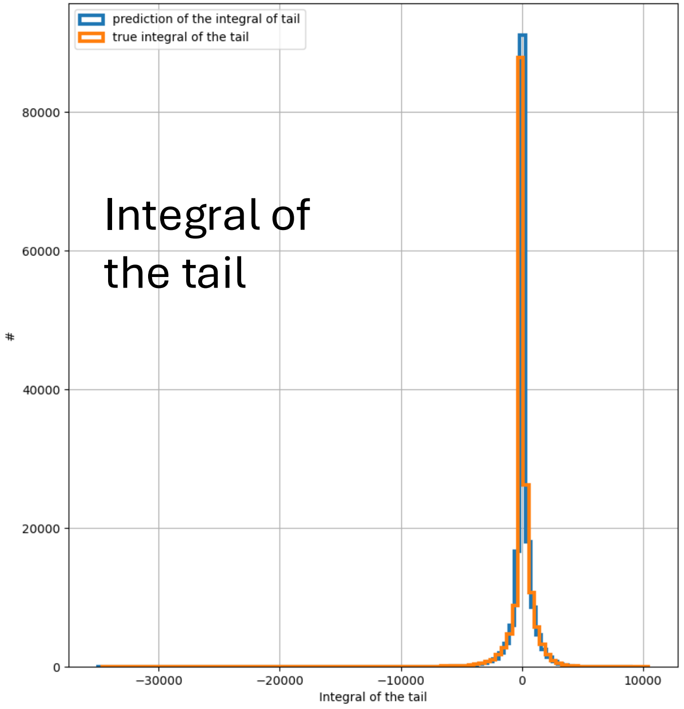
=> Increase the number of samples fed to the models, especially for classes c2 and c3.

Classification: Training on the augmented samples

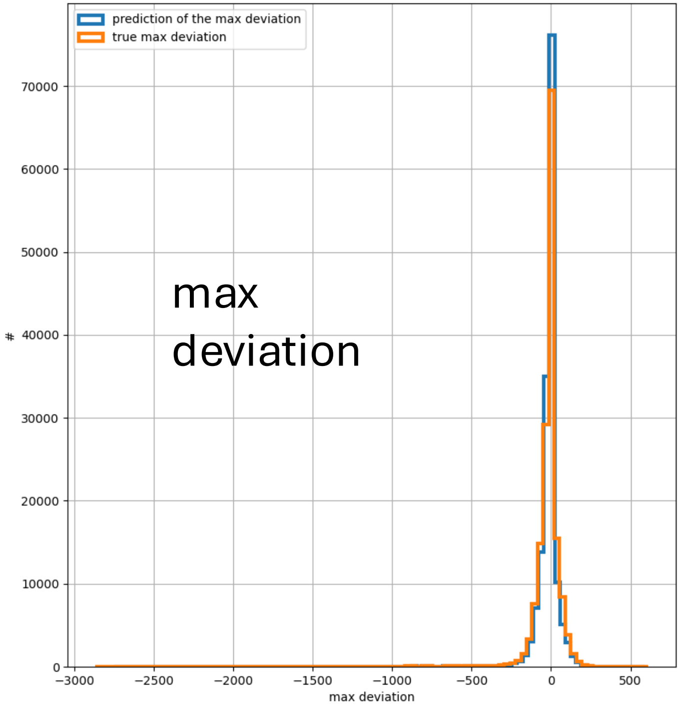
Workflow:
- Train a regressor model to predict the metrics (integral of the tails and max deviation) taking the fit parameters as input.
- Train a classifier model to predict the classes taking the metrics as input.
- Testing the classifier on the predicted integral of tails and max deviation.

Prediction of the integral and max deviation of the testing samples

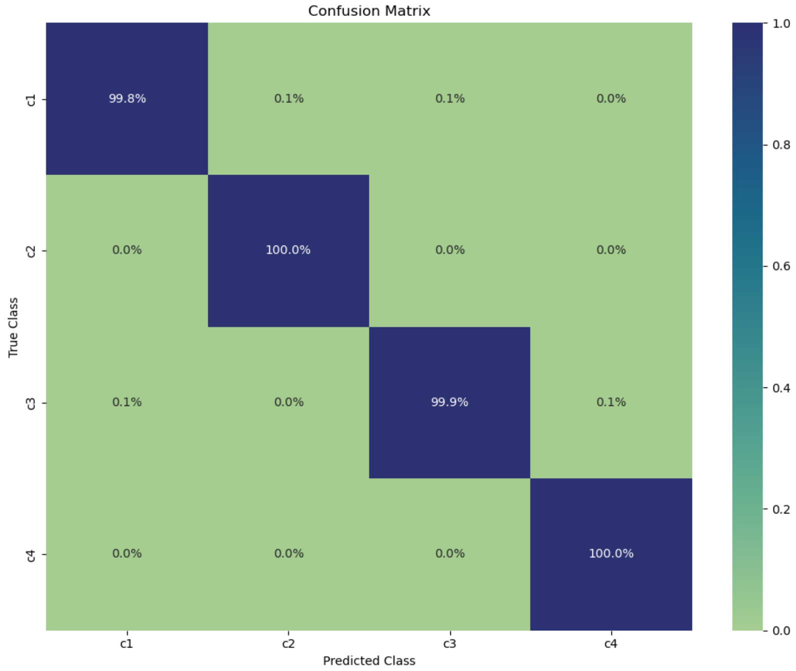
Integral of the tail



max deviation



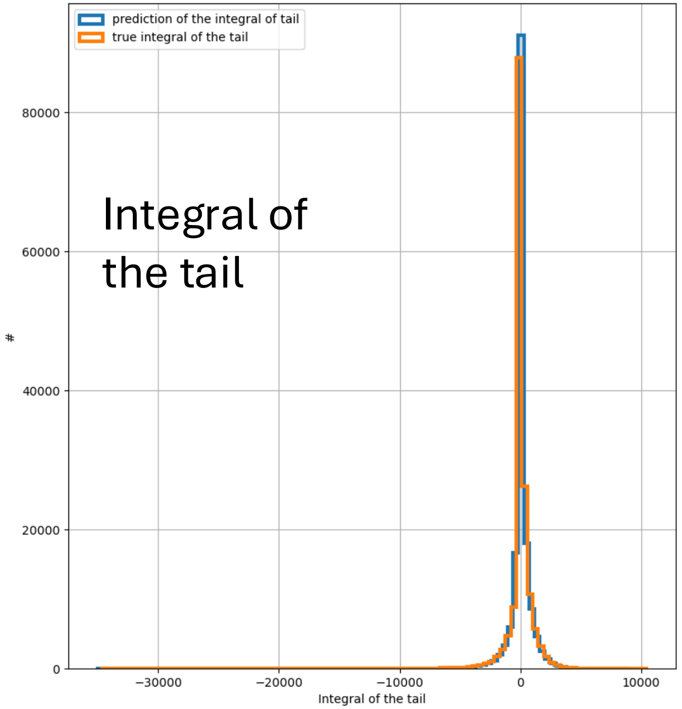
Classification of the testing samples: using the true values of the metrics

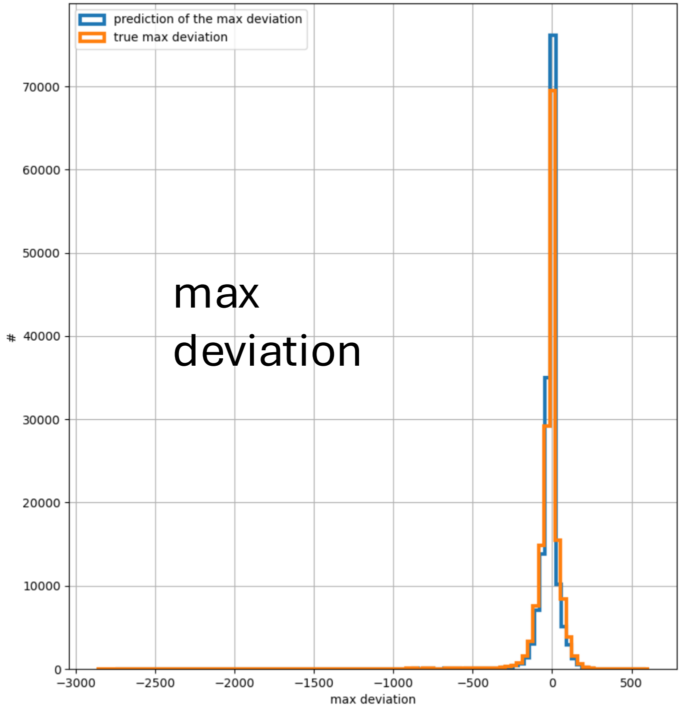


- Discrepancies between the truth and prediction for both metrics are observed.
- The classification of the response using the true values of the metrics yields an accuracy > 99% for each class.

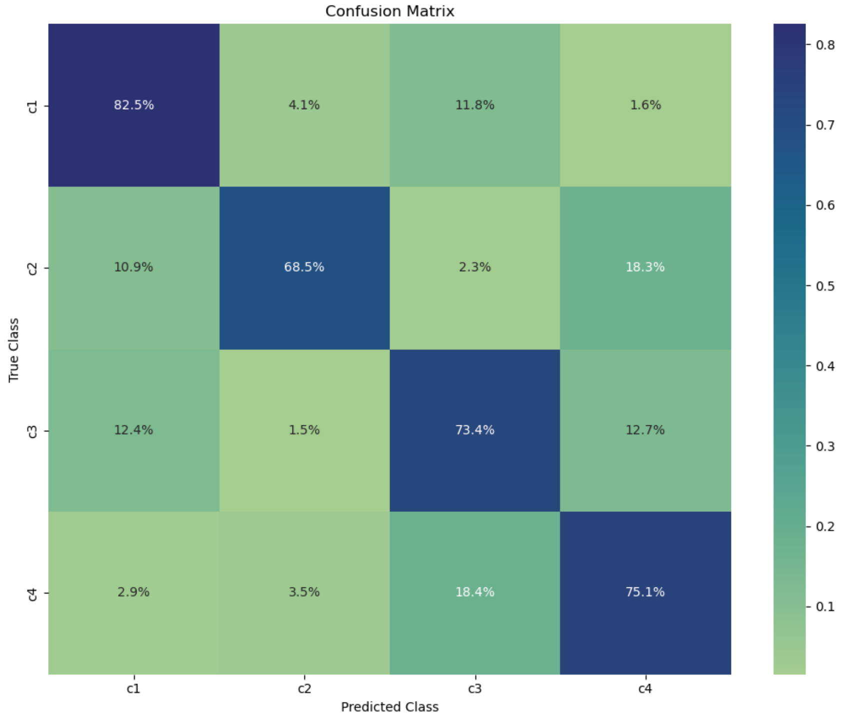
Classification of the predicted integral and max deviation

Input: predicted integral and max deviation of the testing samples





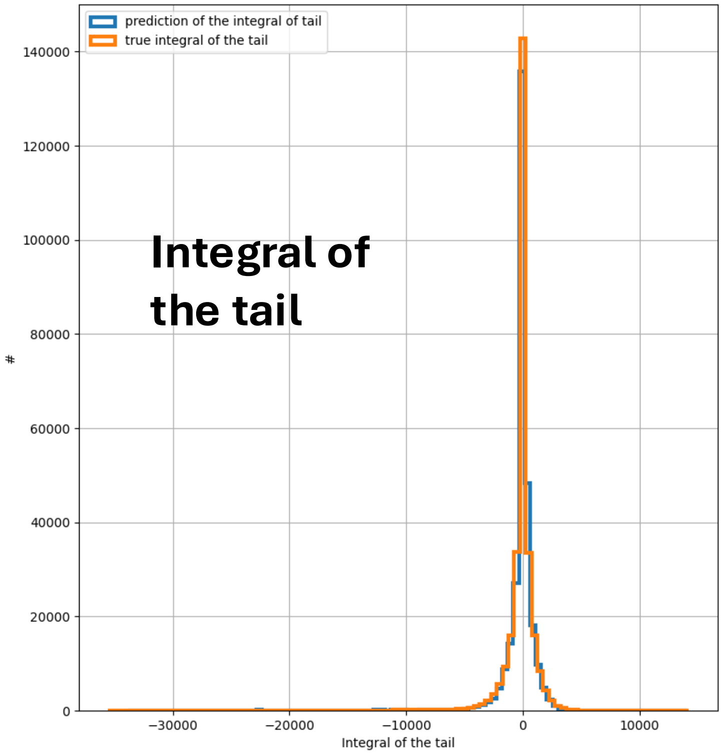
Output

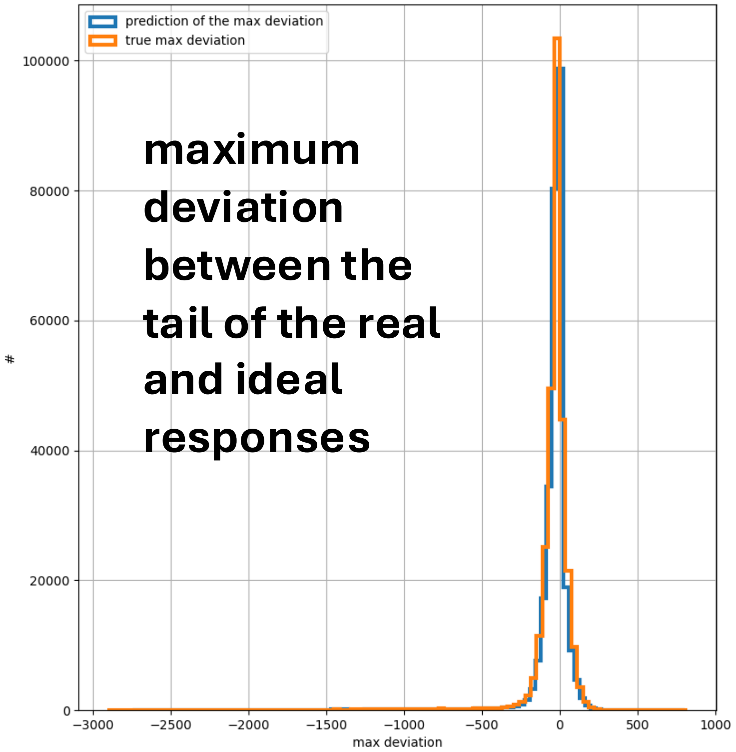


- The overall accuracy of the classification is lower when using the predicted integral of the tails and max deviations.
  - This might be due to imperfect predictions of the integral of the tails and the max deviation.
  - Another possible source of this poor classification is the similarities between c1 and c2, and c3 and c4.
- Comparison with the result of the prediction of true values of the metrics (previous slide) => the BDT representation (used to predict the metrics) of the response function is not as good as the actual one.

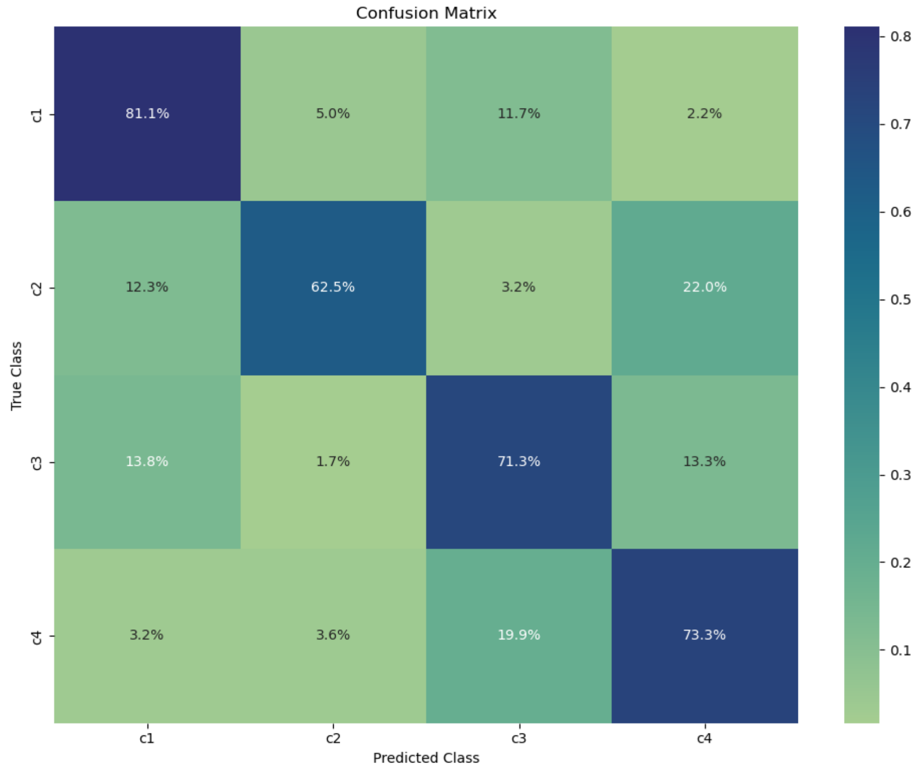
Classification: Testing on the fit parameters from data

Prediction of the integral and max deviation of the data



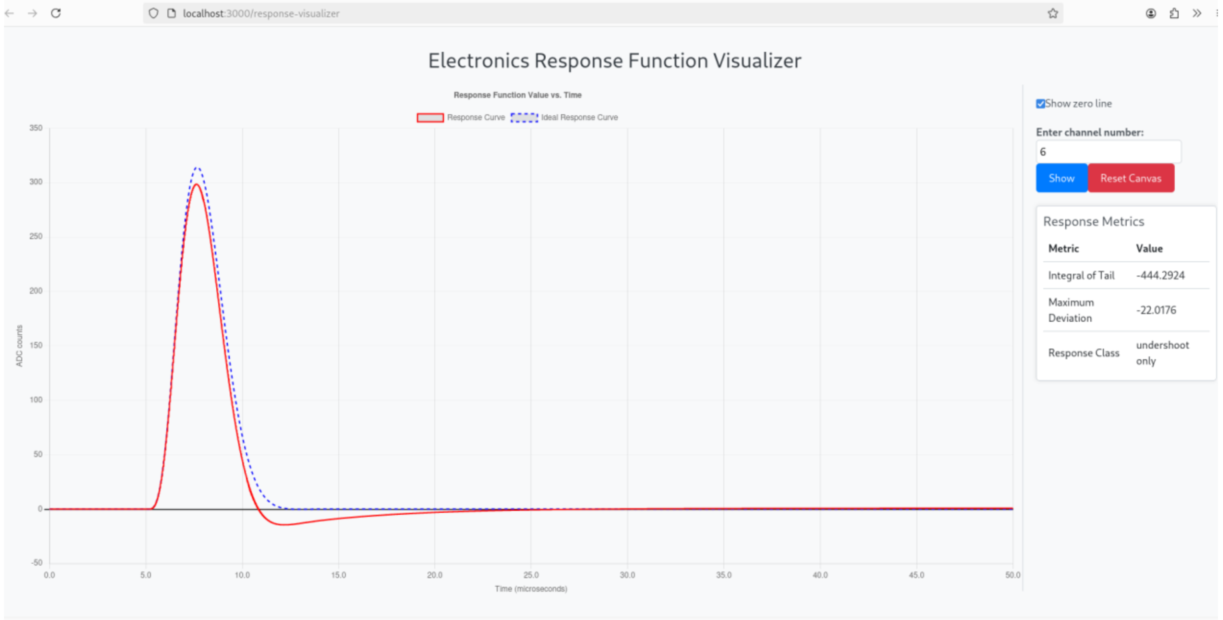


Classification of the predicted integral and max deviation



- Like the prediction using the testing dataset, small discrepancies between the truth and predicted integral of the tails and max deviation are observed.
- As expected, the classification performance on data decreases compared to the testing dataset.
    - The classification of the classes c2 and c3 are a challenge for the BDT.

Visualization



Requirements:

- Back-end:
  - Data in .csv format: in the folder data/
  - BDT models (.json): in the folder models/
  - Conda environment: installed from environment.yml
- Front-end: developed from the elec-response visualizer by Chao and Karla
  - npm to run the application
  - ReactJS is used.

How to run ?

- Within the branch "api_dev" of the repository "CE_Project", run the script api/wsgi.py
- In the repository "elec-response" (branch main), run the app with "npm start".

=> The front-end app will send/retrieve the data from the api.

The chi2 and K-S test needs to be implemented in this web interface.

Back-end: https://github.com/fanrado/CHN_CLASSIFIER_CE/tree/api_dev
Front-end: https://github.com/fanrado/elec-response

Summary

- DUNE's ~1M channels need robust electronics calibration; a memory issue might appear if storing the full waveforms.
- A machine learning-based classification using XGBoost + KDE has been developed to enable parameter-based storage and guide robust fitting.
- An overall accuracy of 97% of the pre-classification on the testing data was achieved.
- A web visualization has been developed to validate the workflow.
- Next steps: implement chi-square, K-S statistical tests, and demonstrate the integration impact on the fit convergence.