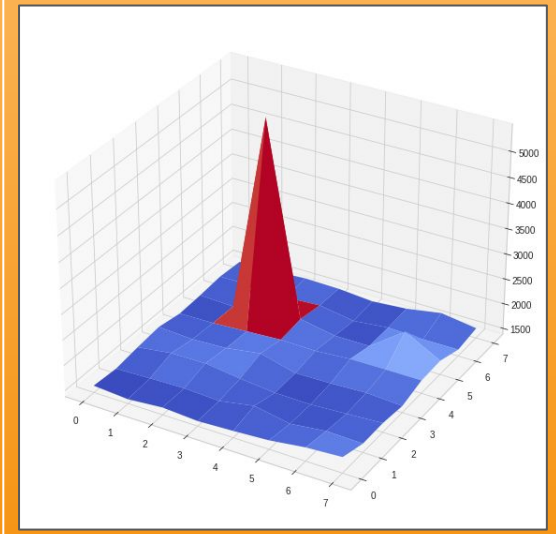
Application of ML to Calorimeter Signal Processing in SPHENIX

Maxim Potekhin

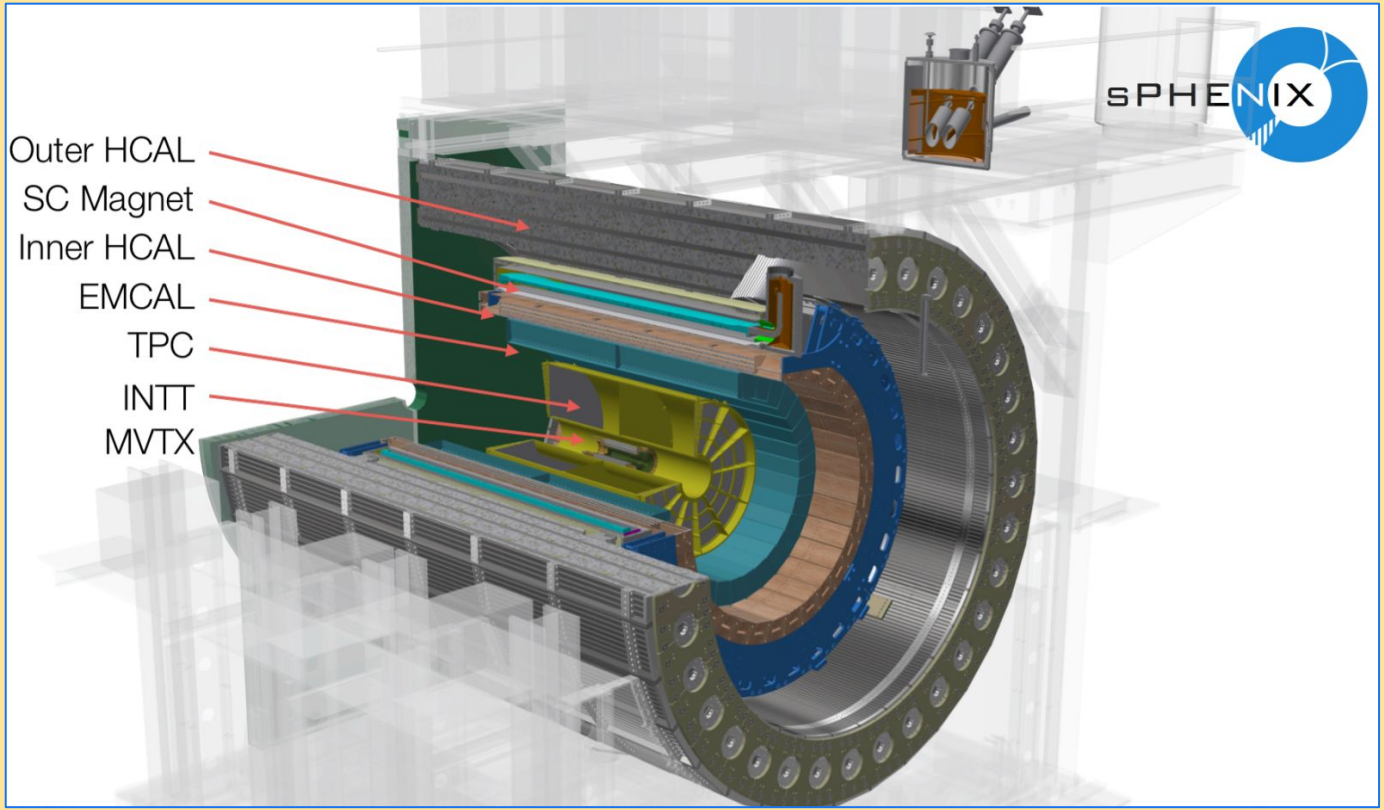
Brookhaven National Laboratory

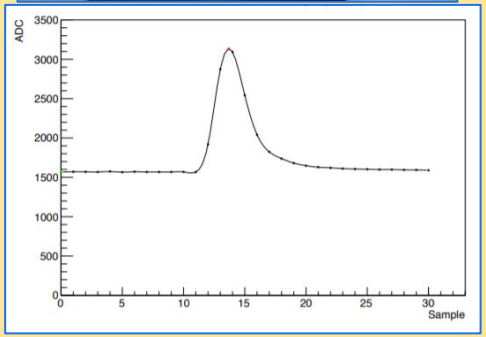
NPPS Group Meeting Meeting

06/10/2022



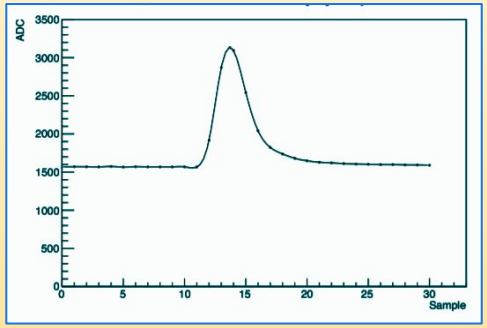
The sPHENIX Calorimeters





SiPM readout of scintillating fibers

EMCal signal processing in a nutshell



?

Your software here

- Peak Amplitude
- Time
- Pedestal

- The peak is a measure of energy, and the signal timing is necessary to build the event
- This applies to EMCal and both HCals in sPHENIX (NB. different architecture)
- 24,576 channels in the EMCal
- ADC Clock is 60MHz
- The input is a vector of integers: 32 in the 2018 test beam experiment, 12 in sPHENIX
- *Effectively, a vector of 12 integers is transformed into a vector of 3 floats (signal features)*

The transformation (a.k.a. signal feature extraction)

- The “standard” method involves fitting the waveform with a parameterized function that represents the signal shape well. The shape is defined by the circuit, transmission lines etc.
- *The parameters* of the fit are then used to determine the signal features.
- There are multiple ways to fit
  - Analytical functions of varying complexity
  - “Signal template” i.e. parameterization of an averaged shape of the signal… etc
- There are other methods of signal processing in existence. Fitting the waveform is currently “mainstream” in sPHENIX (and it works). However, performance may be a problem.

Computing cost

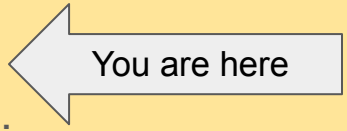
- Using the standard fitting method, it can take between 1.5 and 4 seconds to process an EMCal event for this one stage in the reconstruction chain, depending on zero suppression and other factors. This is a significant fraction of the CPU budget which is ~25s
- …i.e. just the initial peak finding can take up to 16% of the total budget — for just one calorimeter subsystem out of 3
- This is the main motivation to find faster methods of signal feature extraction

“Standard fit” approach vs ML + speed considerations

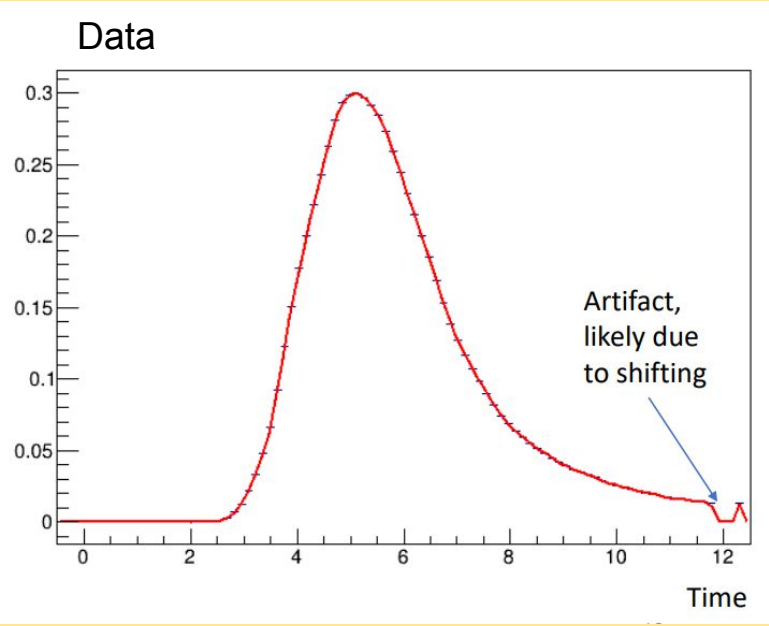
- Standard (credits: T.Rinn, BNL)
  - Apply one of the optimal fitting methods
  - Use multithreading or similar techniques to speed up calculations
- ML
  - Train a model using one of the optimal fitting methods, with the vector representing the signal shape as input
  - Extract the signal features (e.g. the three numbers as explained above) by *inference*
  - Inference is inherently faster than the fit (and can benefit from multithreading as well)
- NB. ML does not aim to be more precise than the traditional fit in this approach – since it is trained to emulate it.
  - However potentially ML can be more flexible for complex feature recognition.

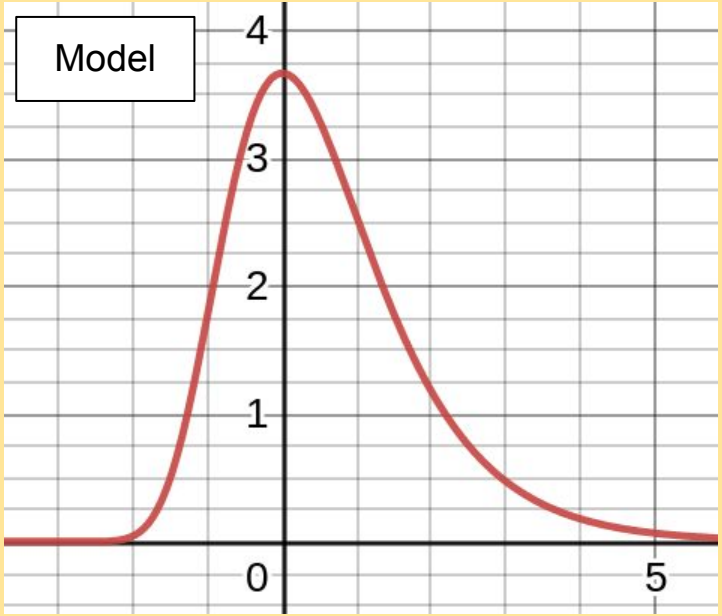
Strategy of this study

- Start with simulated signals – this is a standard practice and helps create a well understood training set.
- Step 1: study just the extraction of the amplitude and timing (toy model).
- Step 2: add noise, time fluctuations and other factors e.g. shape distortions.
- Keep track of ML fit accuracy (residuals distribution, MC truth vs inference).
- Step 3: after the system is understood well enough, move on to real data – test beam data taken in 2018.
- Step 4: deployment and integration design.



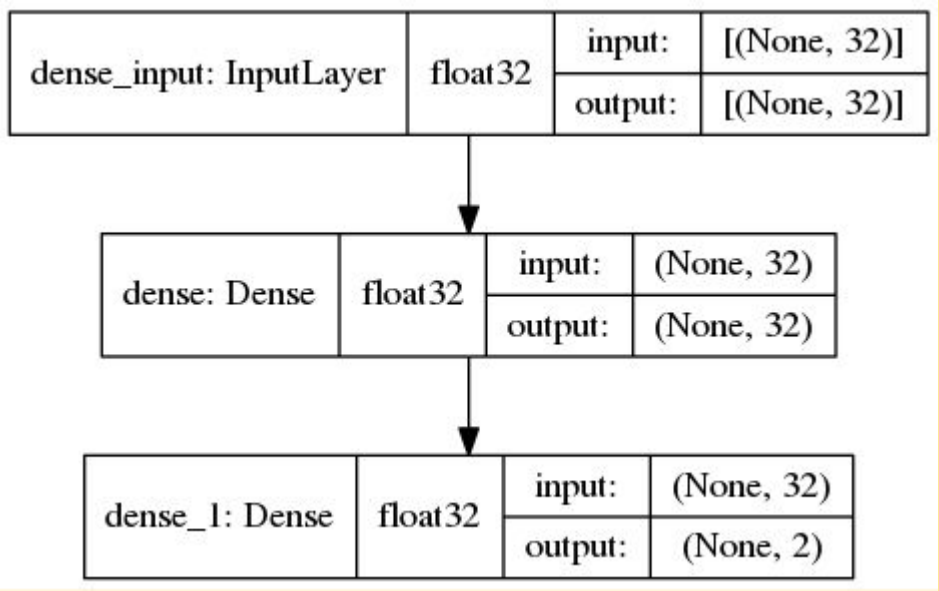
Stage 1: the signal model





- Left: The “real signal” template. Pedestal subtraction was applied.
- Right: Landau distribution. Noise and time fluctuations are added for the ML study.

Initial model: the “Dense” layer in Keras, 2 outputs




```
# The popular “adam” optimizer and the “mse” loss function
model = Sequential()
model.add(Dense(L, input_dim=L, activation='relu'))
model.add(Dense(2, activation=’linear’))
model.compile(loss=’mse’, optimizer=’adam’,
metrics=['accuracy'])

# Inference requires just one function call
answer = model.predict(X, batch_size=batch)
```

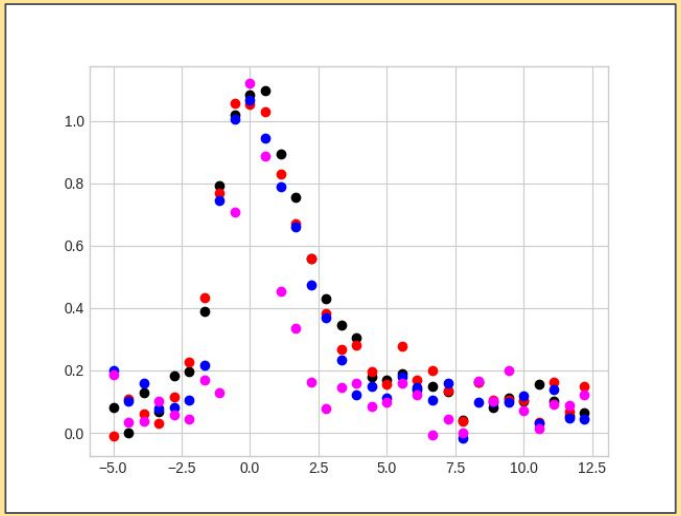
- 32 bins on the “time” axis – according to the test beam conditions (ca. 2018)
- One inner layer
- Two outputs, timing and the amplitude (the pedestal was normalized out)

Summary of the initial study

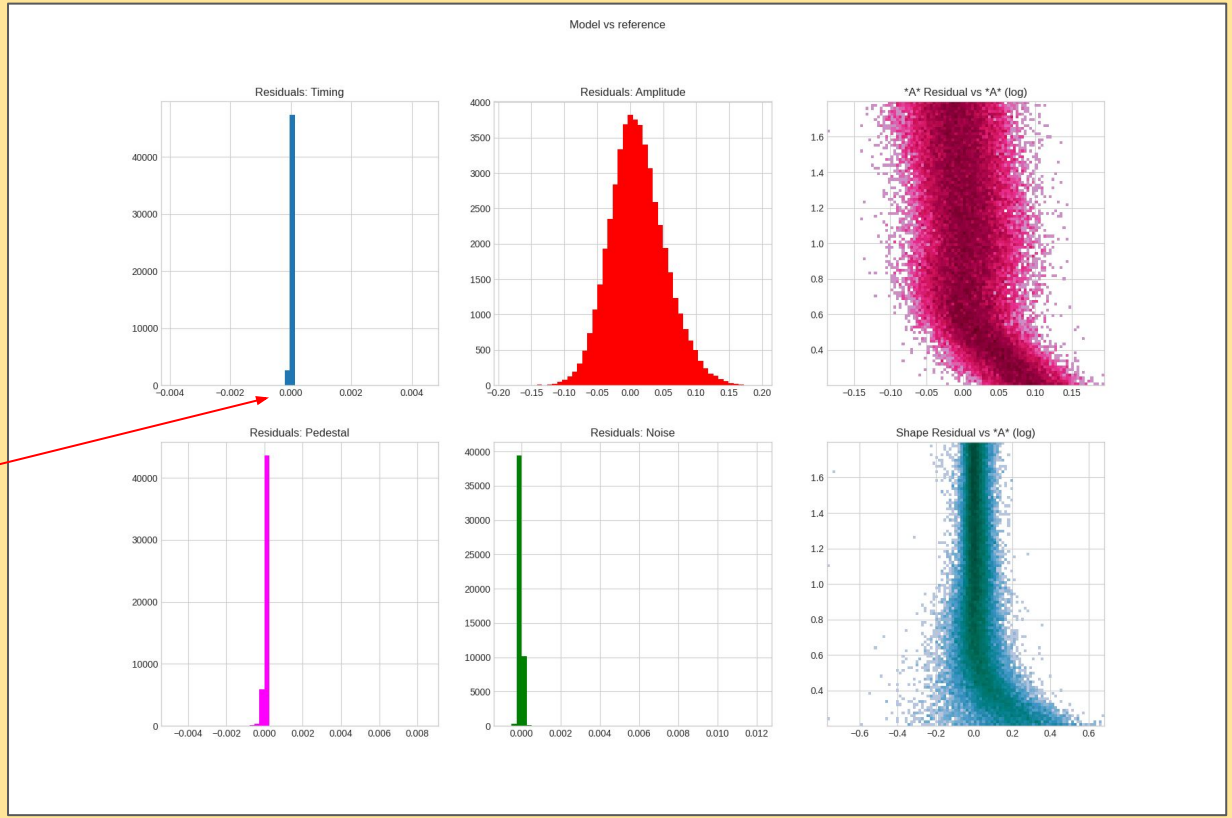
- The “inference batch size” does have an effect on performance (the larger batch the better)
- O(10) times faster than “simple fitting”
  - From ~150μs to less than 10μs
  - Caveats (of course) but an encouraging fist result
- The precision (judged by the standard deviation of the residuals) is adequate
- On to Step 2 (simulated data): add the pedestal as an output parameter, increase noise level to more realistic values, add shape fluctuations

Step 2

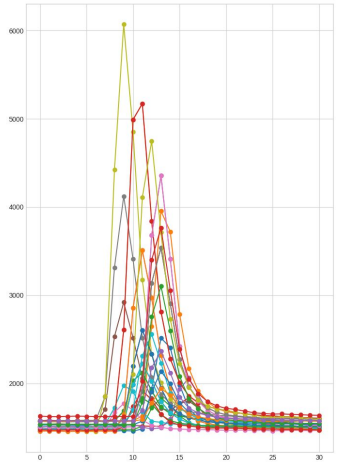
- Added more smearing parameters to the model
- The updated model has five outputs: time, amplitude, pedestal, noise, shape
  - In order to account for more features of the real signal – initially time and amplitude
  - The output layer of the network nas now 5 elements
  - i.e. we are extracting more information from the signal shape

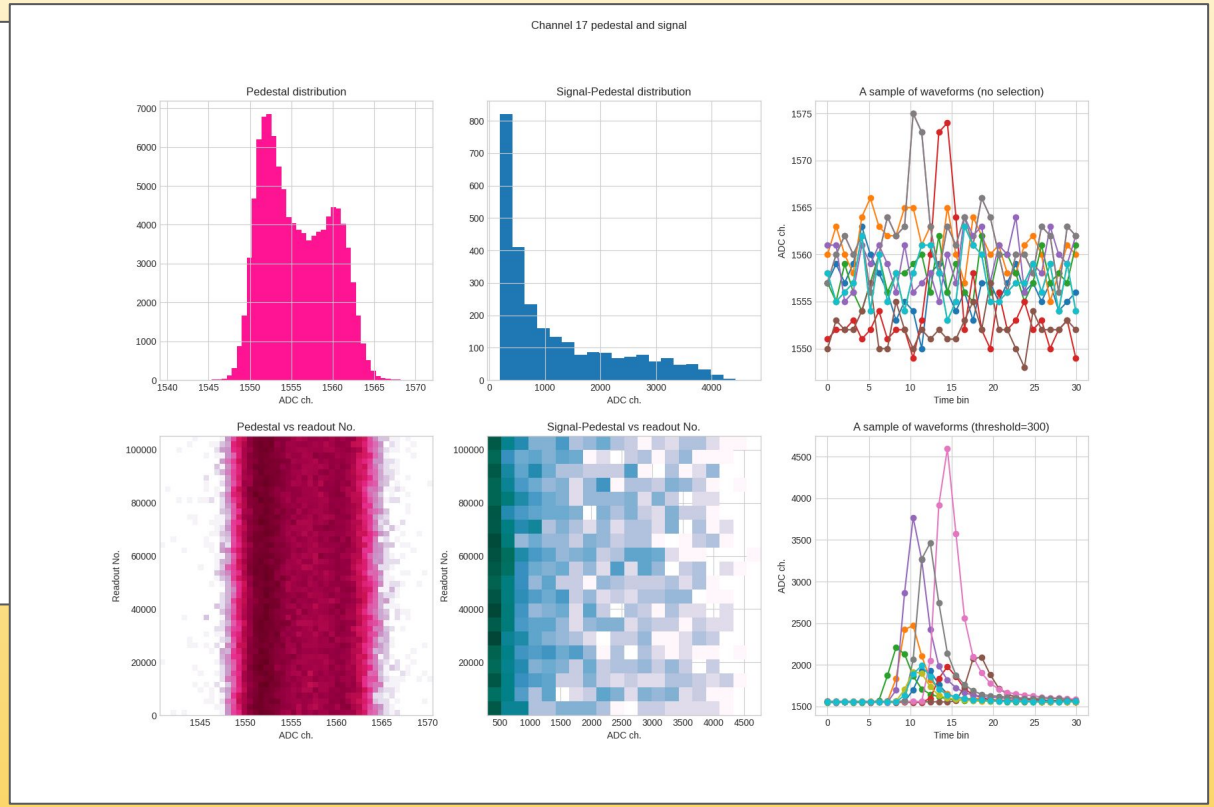


Residuals produced by the inference process (with MC inputs)



Step 3: test beam data – examining basic features

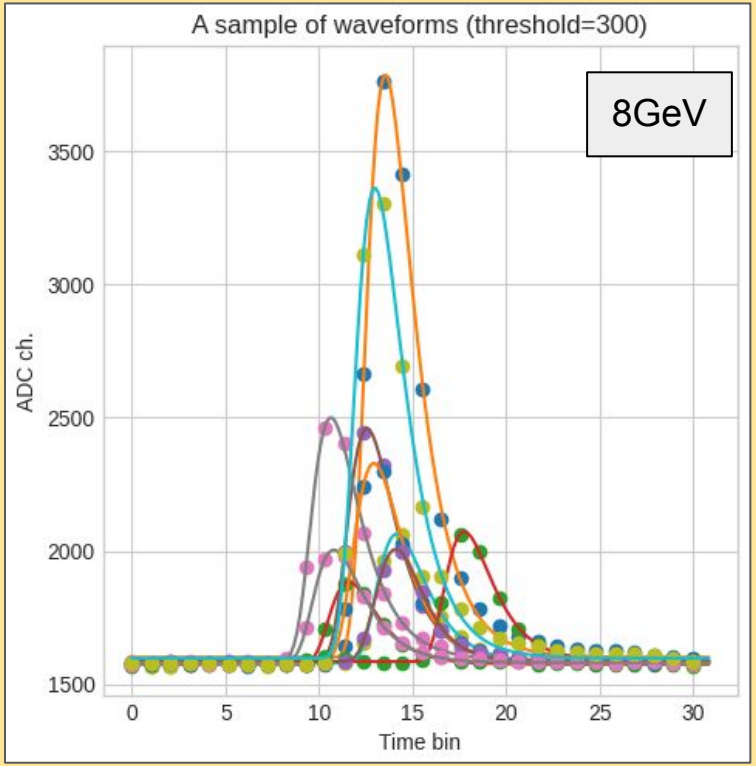




Test run 2018 (FNAL)

Energy Scan
2 – 28 GeV

Fit of the data – for ML training purposes



Define the residuals as $e_i = y_i - f_i$ (forming a vector $\boldsymbol{e}$).

If $\bar{y}$ is the mean of the observed data:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

then the variability of the data set can be measured with two sums of squares formulas:

- The sum of squares of residuals, also called the residual sum of squares:

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

- The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$$

The most general definition of the coefficient of determination is

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

In the best case, the modeled values exactly match the observed values, which results in $SS_{\text{res}} = 0$ and $R^2 = 1$.

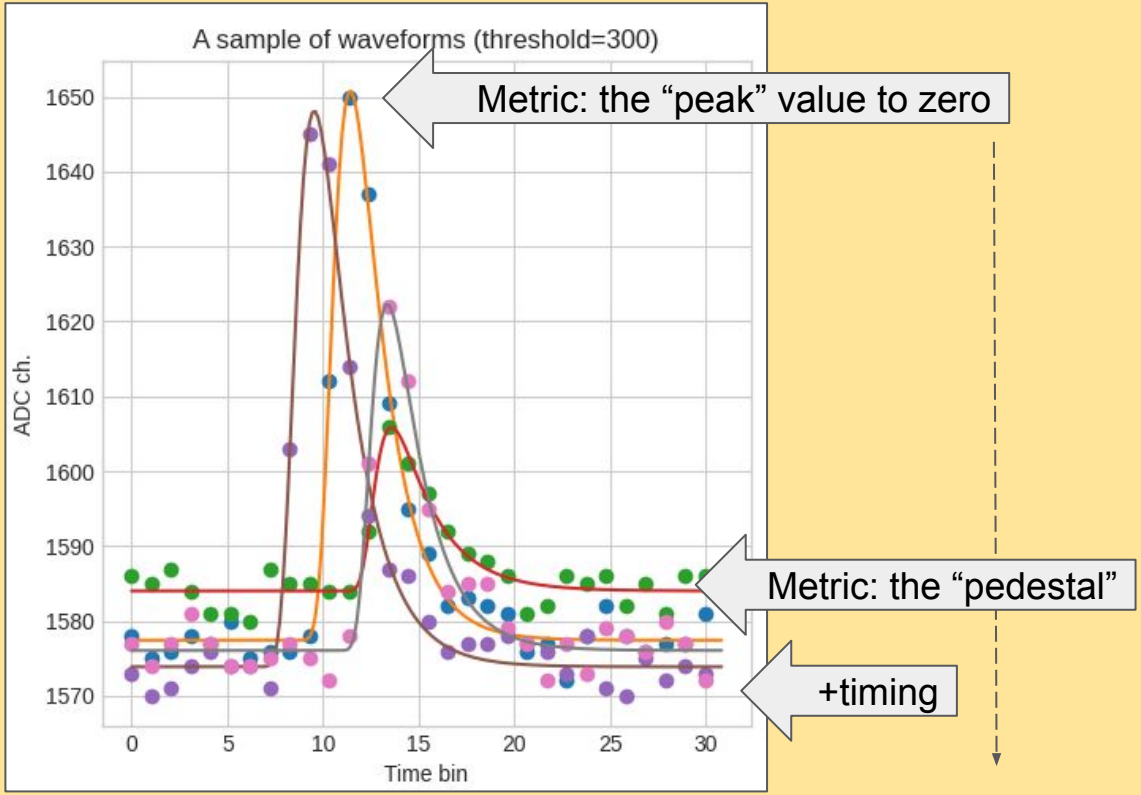
R^2 – goodness of fit

In this case R^2 is > 0.997 in spot checks of 100s of cases – quality is OK

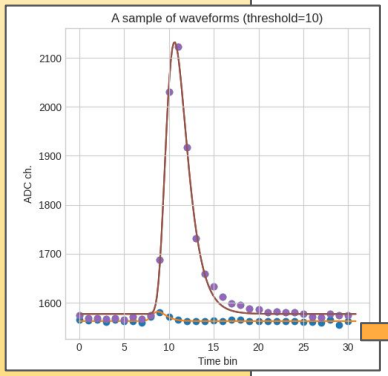
Fitting low-amplitude signals

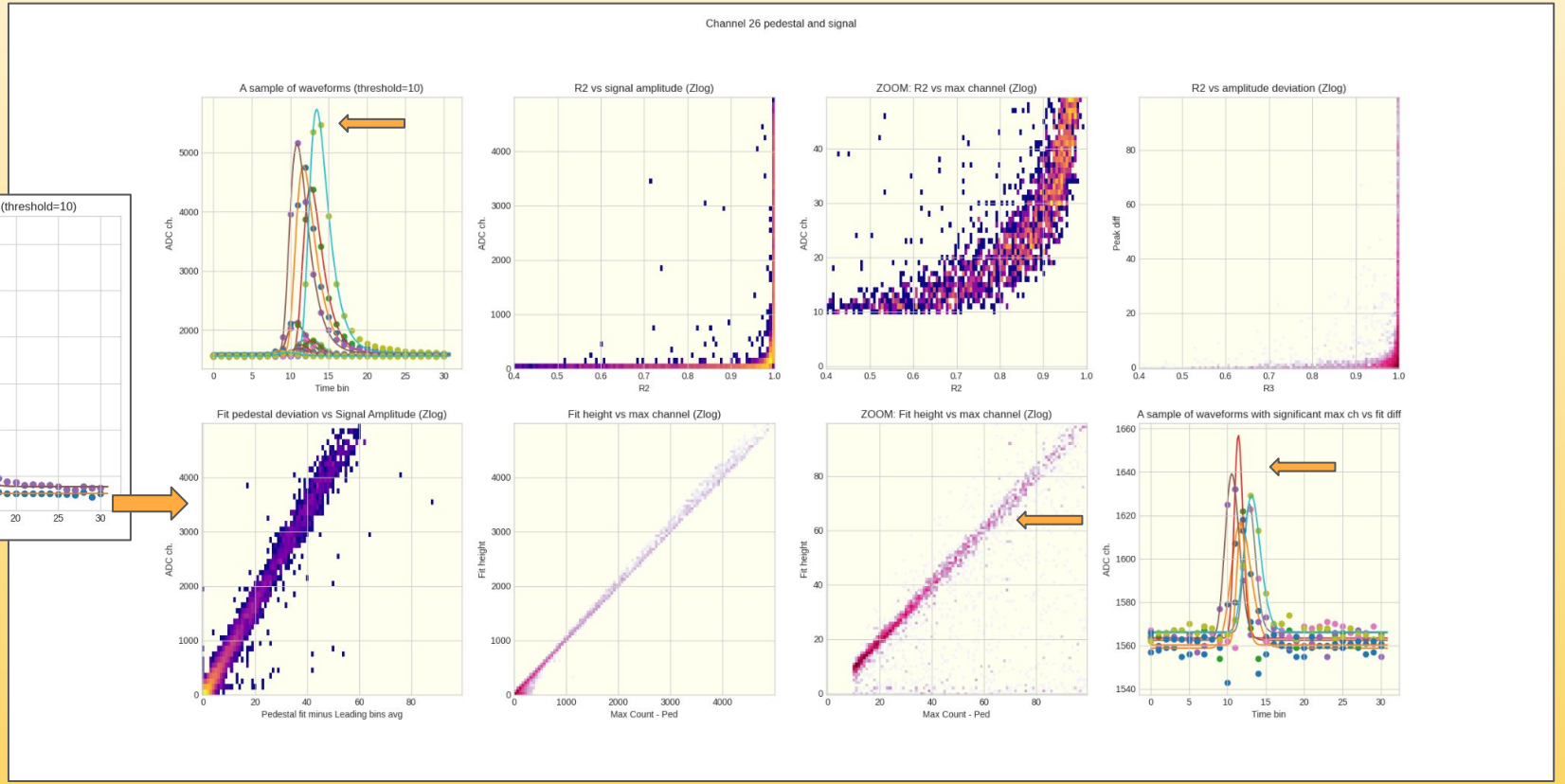
The R^2 metric is used to examine limits of applicability of the fitting procedure (see next slide)

Additionally, fit values of the pedestal are compared with the average over the first 5 bins

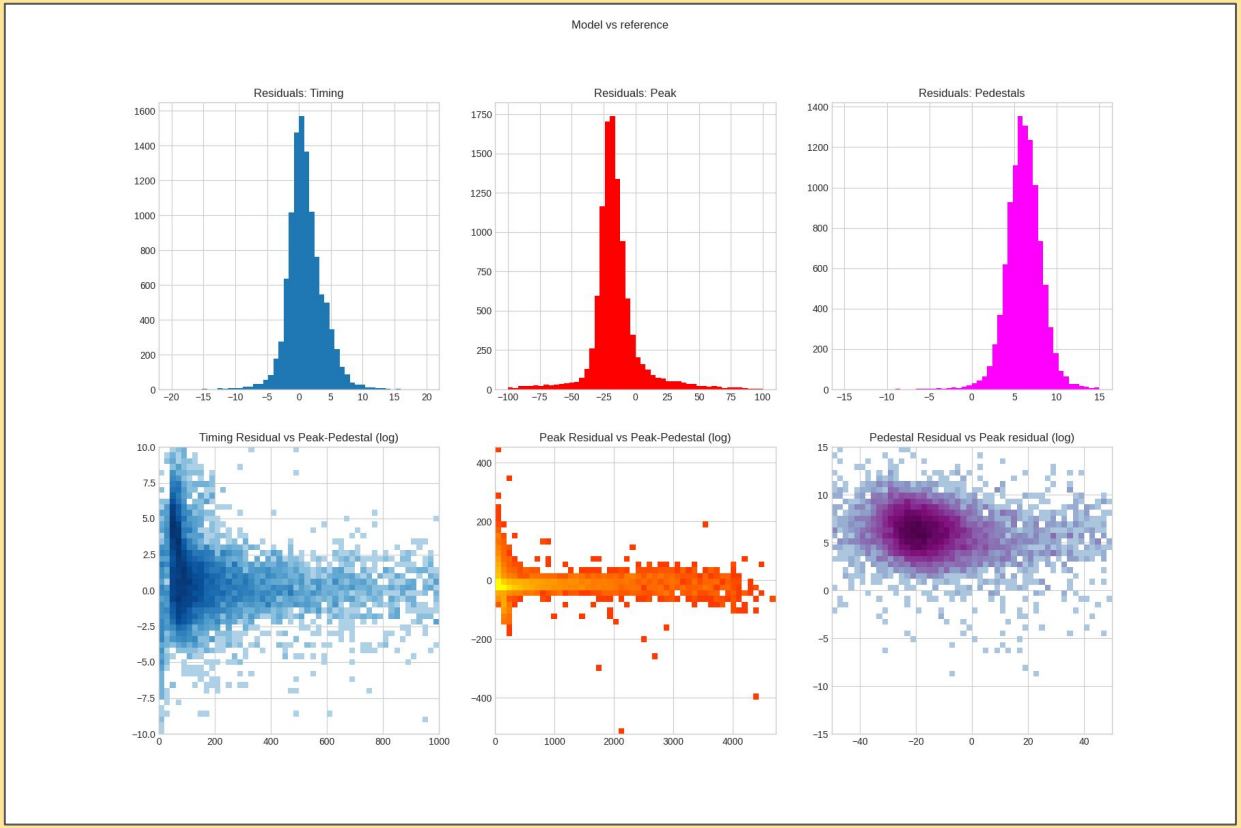


Fit quality checks (including R^2)

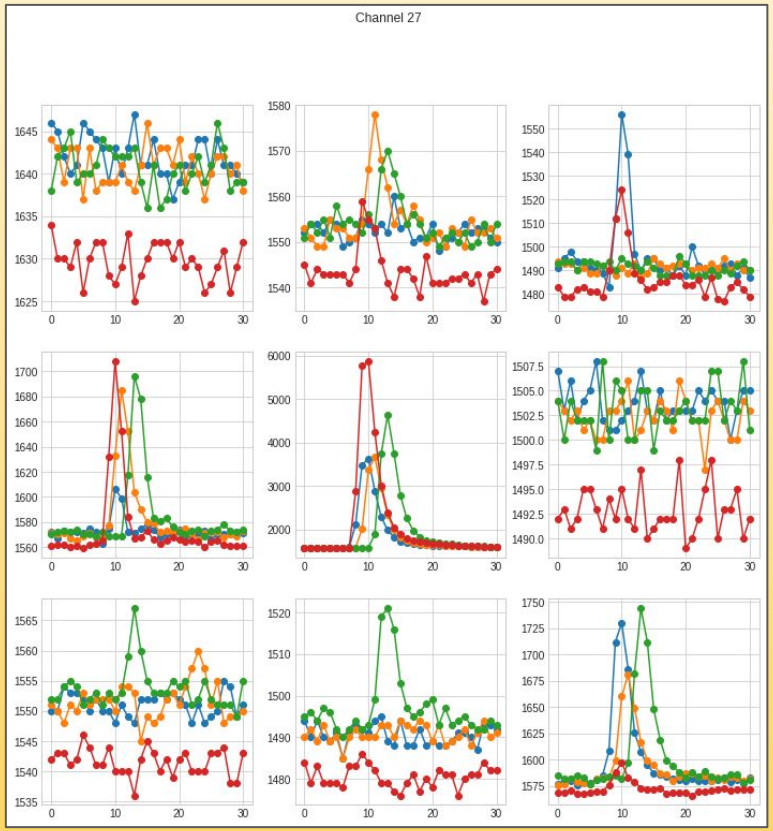




An inference example with real data, 8GeV run



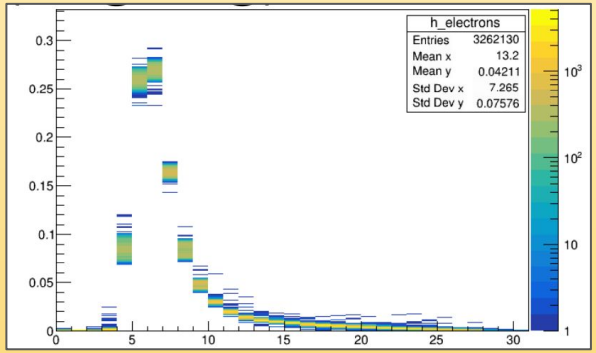
Timing correlations within calorimeter events



- Signals in multiple towers are correlated in time
- Illustrating the concept that a *multi-dimensional fit* should be possible, using timing as a constraint – amenable to ML as well
- Due to uncertainty in the even building strategy in sPHENIX this is currently on hold (cf. separate data streams for the calorimeter sections)

Final notes on the test beam data for ML training

- Currently exploring fit options for the test beam data. Fit quality is important.
- Looked at analytical fit functions, now investigating the “template” method i.e. fitting to a discrete normalized shape of the pulse



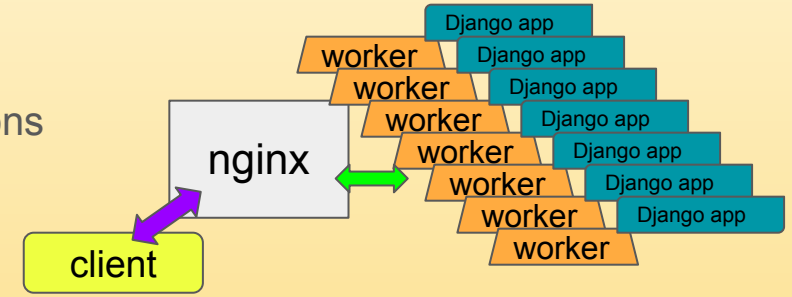
- Current ML software is based on Keras/Python; compatibility with ROOT is achieved via the *“uproot”* package

Step 4 – possible deployment options

- For integration with C++ applications, TF can be compiled from source code using the proprietary “Bazel” tool. Would need more effort and additional integration into the sPHENIX software sack.
- Migration to PyTorch could be possible as well, with easier C++ interfaces.
- If the applications stays in the Python domain, a few options are available:
  - ROOT I/O in Python, using “uproot” or a similar package
  - Inference server (centralized)
  - A lightweight inference service running on the WNs
- The two latter options have been investigated.
- NB. It is possible (and may be desirable) to package all fitting methods in the same service and have complete interoperability. Same API.

Step 4(a): the inference server

- The design is simple and quite common nowadays
- Components:
  - nginx as the front end to handle/sanitize connections
  - gunicorn – the WSGI server (to host Python apps)
  - Django as the back end app platform
  - Keras/TF as the app layer



- Keras models are trained in previous exercises are read directly (once) by Django
- The server receives inputs as binary stream containing NumPy arrays (this can be changed to ROOT if needed)
- Can produce output in various formats needed

nginx-based inference server tested

- 50k channel ML fits per second on a single core (i.e. 20μs per fit)
- Appears to scale with number of cores i.e. 200k fits per second on a 4-core machine
- Batching up input data is important – just like in previous exercises, we observe that time per input is reduced with the increased input size, up to O(10^5) – this has to do with mechanics and optimization in Keras/TF
- Network throughput effects will depend on actual deployment so they were not the focus here
  - The initial test was done with a few machines on a local network switch
  - Deployment results will vary depending on the network topology, performance etc

4(b) – microservice running on the WN

- Considering a more flexible approach where the lightweight service is “planted” on worker nodes – there are multiple options to do this.
- Experimented with long-running Condor jobs, and communicating with those over the network, inside the SDCC perimeter (would be 100% local in real life scenario)
- In this mode, the need for nginx is not compelling and its deployment problematic (can’t do it on the worker nodes), so straight WSGI gunicorn is used
- The service is packaged entirely as a Docker image and deployed on the SDCC farm submitting Singularity jobs to HTCondor
- When running on the same mode, additional (fast) modes of ingesting the data are available e.g. /dev/shm (shared memory)

Summary

- Application of ML methods e.g. Keras/TensorFlow to the calorimeter signal processing has been demonstrated to produce substantial performance gains when compared to traditional fitting methods
- Current work focuses on exploring optimal fitting methods, based on real data from a previous test beam experiment, and on integration with the sPHENIX software stack
- We aim for a common API for a few fitting methods, for efficient evaluation of speed, precision, systematics and deployment
- Two prototype services have been created and tested: a centralized nginx-based inference server, and a lightweight microservice suitable for deployment on the worker nodes