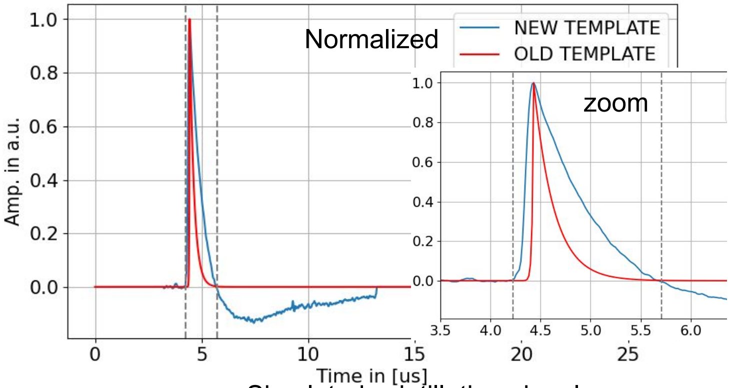
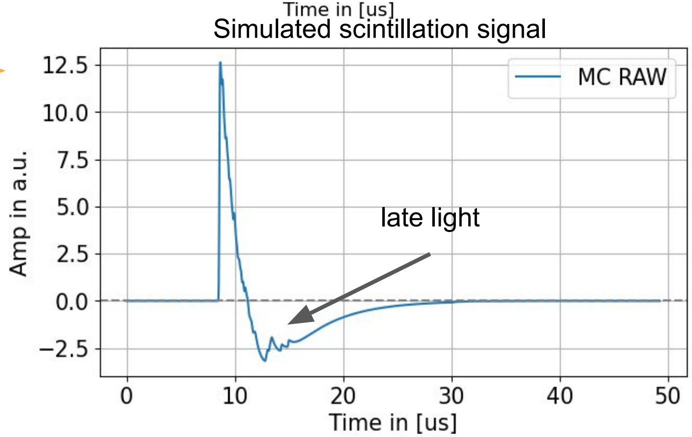
OPHIT FINDER FOR DECONVOLUTION OF REALISTIC SC WVFs IN LARSOFT

Maritza Delgado, Daniele Guffanti, Sergio Manthey Corchado

MOTIVATION

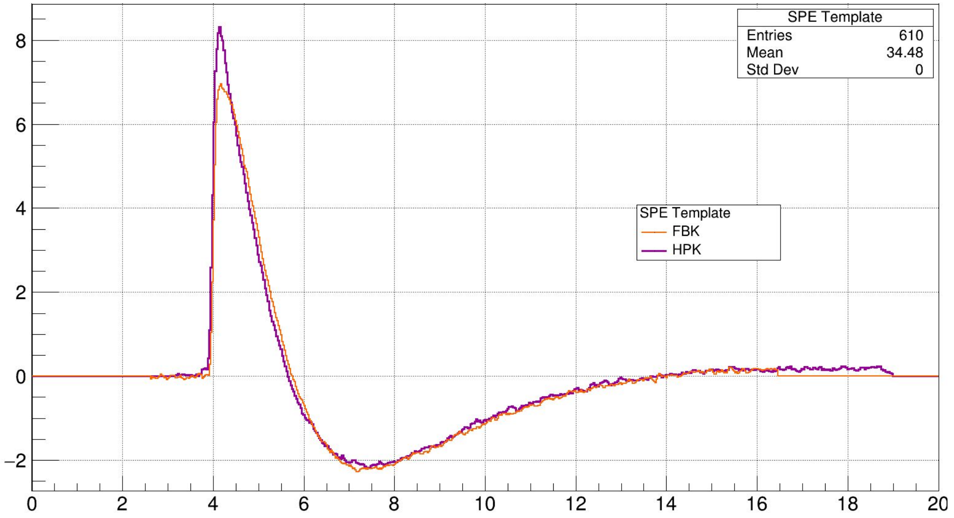
- Implemented **larsoft signals are unrealistic** (OLD TEMPLATE).
- DUNE X-ARAPUCA signals **will have undershoot** (bipolar signals).
- To better **estimate the total charge and time** of each pulse, a deconvolution needs to be implemented.
  - Especially important for **scintillation signals**!
- A deconvolution approach is needed and should provide:
  - Scintillation time profiles.
  - Linear amplitude distribution.
  - Charge and arrival time reconstruction.





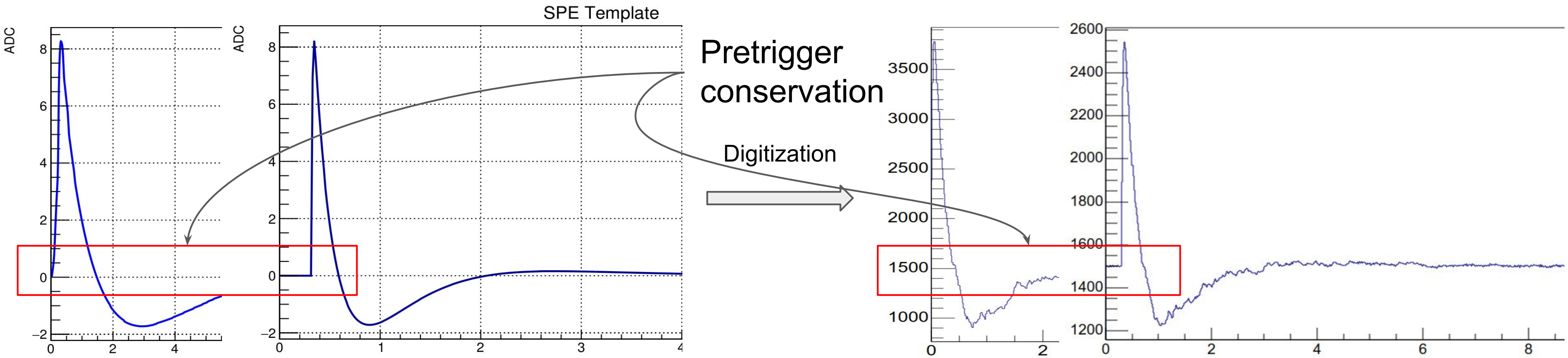
DIGI. WVF REQUIREMENTS

- Digitizer module provides option to include pulse template from version v09_65_00
- SPE template is simulated with DAPHNE V2 and the Cold Amplifier (with 48 SiPM FBK/HPK) enters in both **digitizer** and **deconvolution** modules -> **Provide ideal response**.



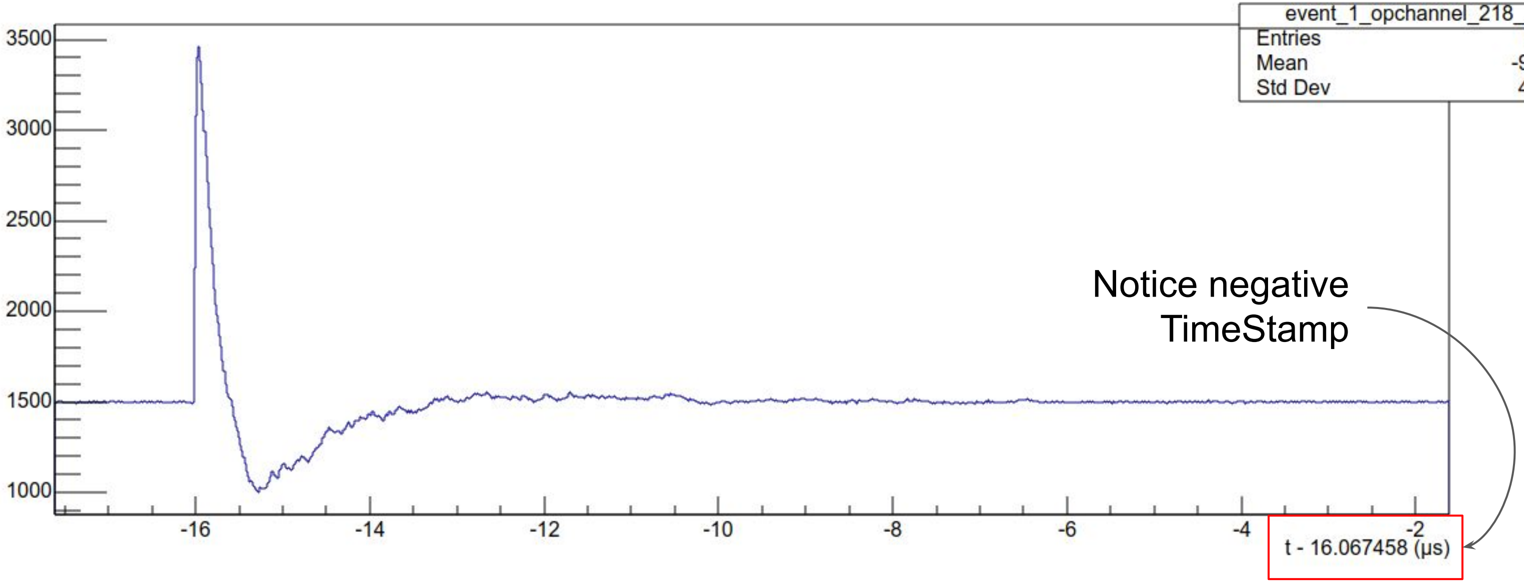
DIGI. WVF REQUIREMENTS

- SPE template is simulated with DAPHNE V2 and the Cold Amplifier (with 48 SiPM FBK) enters in both **digitizer** and **deconvolution** modules -> **Provide ideal response**.
- **From digitization module observed:**
  - ~~if padding = 0 -> Template pretrigger = Digitized wvf pretrigger.~~ But photon arrival times do not coincide with signal peak
  - **else** observed variations in pretrigger lengths (?) and negative TimeStamps.



DIGI. WVF CHALLENGES

- **From digitization analyzer:**
  - **Time axis constructed from lowest TimeStamp in (us).**



WORKFLOW

DIGITIZER FCL

- **Performance test require to switch off dark noise and crosstalk** (this #PE are not stored as true photoelectrons and thus make the analysis more difficult)
- **Pretrigger** and **array length** are adjusted:
  - ~~With padding = 0 pretrigger from template gets passed to raw wvf.~~ Looking for best config.
  - Total window length gets defined with the input parameters of SSP algo:

```python
BEGIN_PROLOG

algo_sspleadingedge:
{
Name: "SSP_LED"
ADCThreshold: 10
Pedestal: 1500
DWindow: 10
ReadoutWd: 1001 # Actual wvf max length in [ticks]
PreTrg: 100
}

END_PROLOG
```

```python
#include "SSPAlgorithm.fcl"
BEGIN_PROLOG

standard_digitizer:
{
module_type: "OpDetDigitizerDUNE"
InputModules: ["largeant"] # Module that created simphotons

VoltageToADC: 151.5 # 18.45 Converting mV to ADC counts
LineNoiseRMS: 1.    # Pedestal RMS in ADC counts, likely an underestimate
DarkNoiseRate: 0.   # (10) In Hz, Ranges 2-50 depending on Vbias 0.996
CrossTalk: 0.       # (0.2) Probability of producing 2 PE for 1 incident photon

Pedestal: 1500 # in ADC counts
FullWaveformOutput: false # Output full waveform.
DefaultSimWindow: false # Use -1*drift window as the start time and
TimeBegin: 0 # In us (not used if DefaultSimWindow is set to true)0.2
TimeEnd: 16 # In us (not used if DefaultSimWindow is set to true)
ReadoutWindow: 1000 # In ticks
PreTrigger: 100 # [ticks] Does not do anything with padding = 0
algo_threshold: @local::algo_sspleadingedge

Padding: 0 # In ticks
PeakTime: 1.960 # 1.950
SSP_LED_DigiTree: true # To create a SSP LED trigger Ttree
SinglePEsignal: true # false for ideal XArapuca response, true for testbench
SPEDataFile: "../template/fbk_digi.txt" # Path to SPE template
TestbenchSinglePE: true # Bool to select SPE template

#Parameters for SiPM-like shape
PulseLength: 16.0 # hpk 7.2 (FBK 0.0095)
MaxAmplitude: 0.04092 # 0.04092 (FBK 0.0095) # * VoltageToADC = 6.2 ADC/PE
FrontTime: 0.097 # 0.013(0.097 FBK)(0.146 )
BackTime: 0.91 # 0.51 (0.676hpk)
}

END_PROLOG
```

DECONVOLUTION FCL

```python
BEGIN_PROLOG

standard_deconvolution:{
module_type: "Deconvolution"
InputModule: "opdigi"
InstanceName: ""

LineNoiseRMS: 3 # Pedestal RMS in [ADC] counts, likely an underestimate
TimeBegin: 0 # In [us]
TimeEnd: 16 # In [us]
PreTrigger: 100 # In [ticks] 25
Pedestal: 1500 # In [ADC]
Samples: 1000 # MaxTimewindow in [ticks]
PedestalBuffer: 20 # In [ticks], should always be smaller than PreTrigger!
Scale: 0.001 # Sacilng of resulting wvfs
DigiDataFile: "../template/fbk_deco.txt"
DigiDataColumn: 0
AutoScale: true # Scaling based on SPE amp. from template

ApplyPhantomPretrigger: false # Add phantom pretrigger
ApplyPrefilter: false # Filter the waveforms before deconvolution
ApplyPostfilter: false # Filter the waveforms after deconvolution
ApplyPostBLCorrection: true # Correct baseline after the deconvolution process

WfmPostfilter: {
Name: "Gauss"
Cutoff: 2.8 # In MHz
}

WfmPrefilter: {
Name: "Gauss"
Cutoff: 2. # In MHz
}

WfmFilter: {
Name: "Wiener"
Cutoff: 1.
}
}
```

Deconvolution module takes as input raw::OpDetWaveform and returns recob::OpWaveform. Some config features include:

- Gaussian noise computed according to RMS [ADC].
- Pedestal [ADC] subtracted before deconvolution.
- Path to SPE template.
- Module structured to provide a 3-step processing:
  - **New Postfilter config!**: Currently only "Gauss" filter implemented.
  - Prefilter config: Currently only "Gauss" filter implemented.
  - Filter config: Currently "Gauss" and "Wiener" are implemented. Frequency cutoff can be adjusted to change "strength" of gauss filter.
- Boolean variables can be used to choose filter combinations and scaling options and more.

DECONVOLUTION MODULE

- **Pretrigger** and **array length** are adjusted.
- Module performers the fft of signal, SPE and noise.
- The module proposes three deconvolution approaches:
  - Wiener Filter:

$$G(f) = \frac{1}{H(f)}\left[\frac{1}{1+1/\left(|H(f)|^2 \mathrm{SNR}(f)\right)}\right]$$

$$\mathrm{SNR}(f) = \frac{|S(f)|^2}{|N(f)|^2}$$

  - Gauss Filter:

$$G(f) = \begin{cases} \exp\left\{-\frac{1}{2}\left(\frac{f}{f_c}\right)^2\right\} & f > 0 \\ 0 & f = 0 \end{cases} \times \frac{1}{H(f)}$$

  - No filter.

$$G(f) = \frac{1}{H(f)}$$

```
C++
//*******************************
// Compute filters.
//*******************************

for (int i=0; i<fSamples*0.5+1; i++) {
// Compute spectral density

double H2 = xH.fCmplx.at(i).Rho2();
double S2 = xS.fCmplx.at(i).Rho2();
double N2 = fLineNoiseRMS * fLineNoiseRMS * fSamples ;

if (fApplyPrefilter) {
Double_t prefilter_PSD = xG0.fCmplx.at(i).Rho2();
N2 *= prefilter_PSD;
}
```

```
if (fFilterConfig.fType == Deconvolution::kWiener){
// Compute Wiener filter
xG.fCmplx.at(i) = TComplex::Conjugate(xH.fCmplx.at(i))*S2 / (H2*S2 + N2);
}
```

```
else if (fFilterConfig.fType == Deconvolution::kGauss){
// Compute gauss filter
Double_t gauss_cutoff = fFilterConfig.fCutoff;
xG.fCmplx[0] = TComplex(0,0);
xG.fCmplx.at(i) = TComplex::Exp(
-0.5*TMath::Power(i*1e-6*fSampleFreq/(fSamples*gauss_cutoff),2))
/xH.fCmplx.at(i);
}
```

```
else{
// Compute dec signal
xG.fCmplx.at(i) = TComplex::Power(xH.fCmplx.at(i),-1);// Standard dec is just the division of signal and SPE template in Fourier space
}
```

https://github.com/mjdelgadog/Deconvolution.git

OPHITFINDER MODULE

```
namespace opdet {

  void RunHitFinder(std::vector<raw::OpDetWaveform> const&,
                    std::vector<recob::OpHit>&,
                    pmtana::PulseRecoManager const&,
                    pmtana::PMTPulseRecoBase const&,
                    geo::GeometryCore const&,
                    float,
                    detinfo::DetectorClocksData const&,
                    calib::IPhotonCalibrator const&,
                    bool use_start_time = false);

  void RunHitFinder_deco(std::vector<recob::OpWaveform> const&,
                    std::vector<recob::OpHit>&,
                    pmtana::PulseRecoManager const&,
                    pmtana::PMTPulseRecoBase const&,
                    geo::GeometryCore const&,
                    float,
                    detinfo::DetectorClocksData const&,
                    calib::IPhotonCalibrator const&,
                    bool use_start_time = false);

  void ConstructHit(float,
                    int,
                    double,
                    pmtana::pulse_param const&,
                    std::vector<recob::OpHit>&,
                    detinfo::DetectorClocksData const&,
                    calib::IPhotonCalibrator const&,
                    bool use_start_time = false);

} // End opdet namespace
```

- The product recob::OpWaveform was included in the algorithms used by OpHitFinder to produce optical hits: OpHitAlg.cxx, OpHitAlg.h.
- The RunHitFinder_deco function was created to make analysis independent of raw::OpDetWaveform.
- Currently it is possible to use this algorithm with the two products raw and recob.

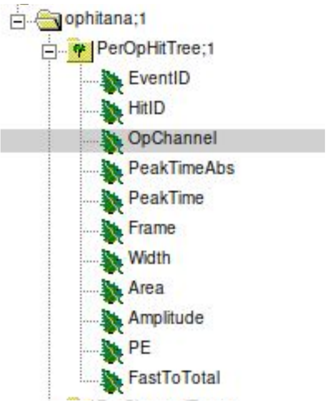
OPHITFINDER MODULE

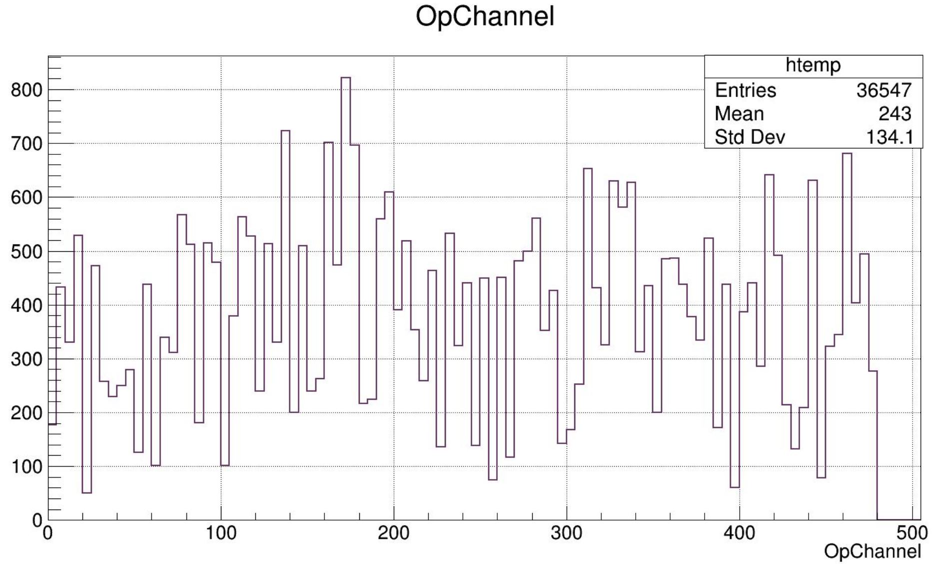
```
//------------------------------------------------------------------------------
void RunHitFinder_deco(std::vector<recob::OpWaveform>const& opWaveformVector,
                  std::vector<recob::OpHit>& hitVector,
                  pmtana::PulseRecoManager const& pulseRecoMgr,
                  pmtana::PMTPulseRecoBase const& threshAlg,
                  geo::GeometryCore const& geometry,
                  float hitThreshold,
                  detinfo::DetectorClocksData const& clocksData,
                  calib::IPhotonCalibrator const& calibrator,
                  bool use_start_time)
{

  for (auto const& deco_waveform : opWaveformVector) {

    const int channel = static_cast<int>(deco_waveform.Channel());

    if (!geometry.IsValidOpChannel(channel)) {
      mf::LogError("OpHitFinder")
        << "Error! unrecognized channel number " << channel << ". Ignoring pulse";
      continue;
    }

    std::vector<short int> short_deco_waveform;
    for (unsigned int i_tick=0; i_tick < deco_waveform.Signal().size(); ++i_tick)
    {
    short_deco_waveform.emplace_back(static_cast<short int>(deco_waveform.Signal().at(i_tick)));
    }

    pulseRecoMgr.Reconstruct(short_deco_waveform);
    //pulseRecoMgr.Reconstruct(deco_waveform);

    // Get the result
    auto const& pulses = threshAlg.GetPulses();

    double timeStamp = double (deco_waveform.TimeStamp().GetTimeStamp());
```

- In the RunHitFinder_deco function:
- PulseRecoManager.cxx: raw Waveform is defined as a short vector, a loop was included to convert the std::vector<float> OpWaveform::Signal() to short. So far it works (with a corresponding sacrifice in resolution).
- ConstructHit calls a double TimeStamp, but after conversion we get unsigned integers (not suited for this purpose). See slide 23.

OPHITFINDER MODULE

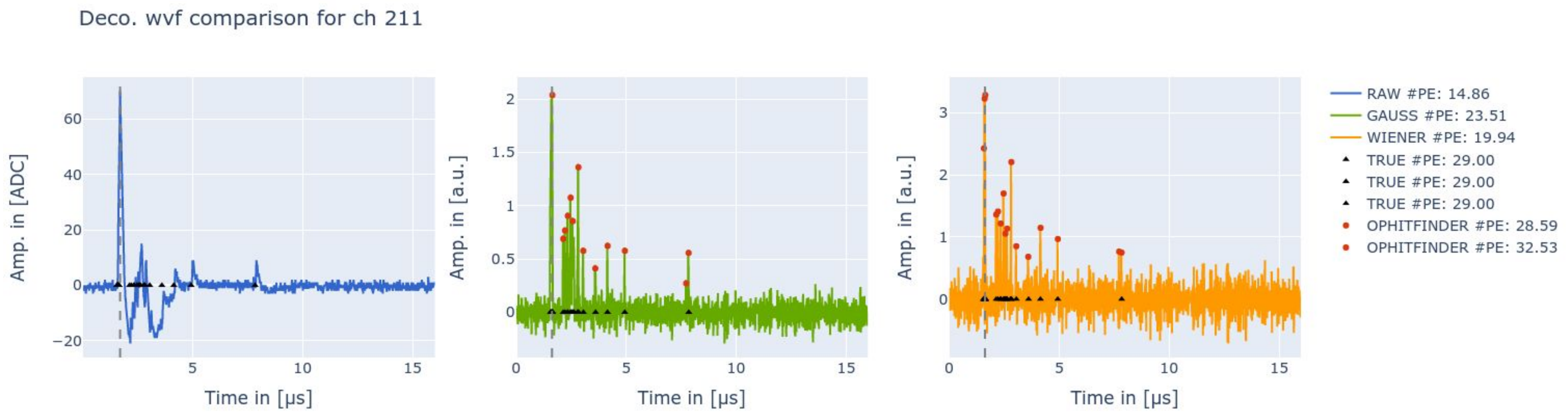
But despite the drawbacks, we obtain from ophitfinder the number of hits per channel and the analysis of the deconvoluted signals has been carried out.





MODULE OUTPUTS

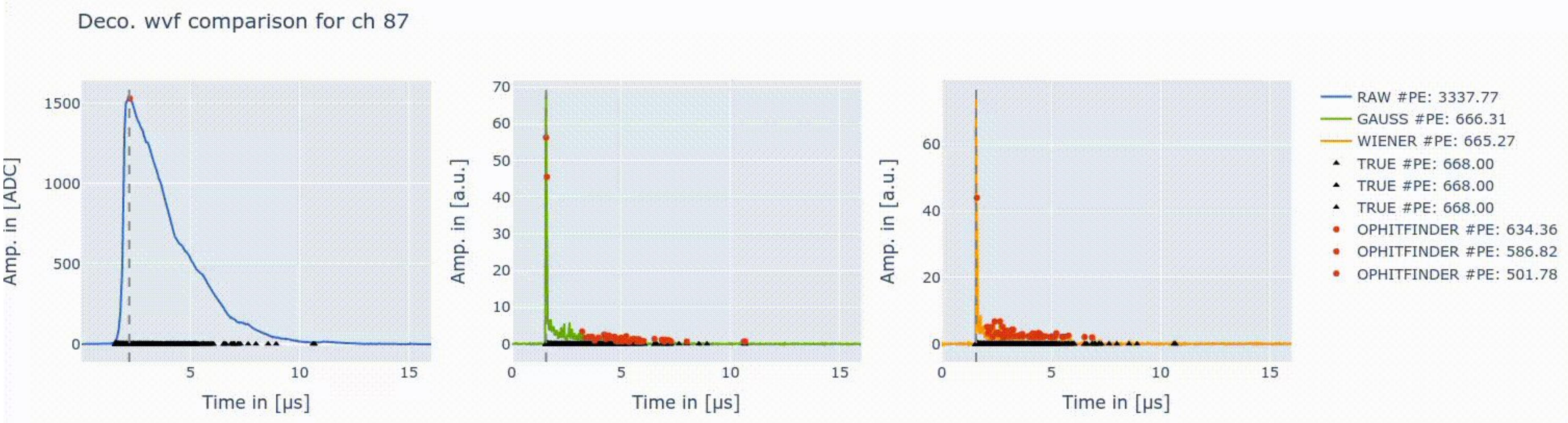
- Resulting dec. waveforms are returned in **units of SPE template**. Regular amp*time integration does not convert to original charge but **this fact is already accounted for in OpHitFinder module**.
- An **automatic rescaling** according to SPE amp. can be applied or imputed from the fcl config. after a **calibration process**.
  - Best scaling strategy depends on ophit finder algorithm configuration. Currently being studied!



ANALYSIS

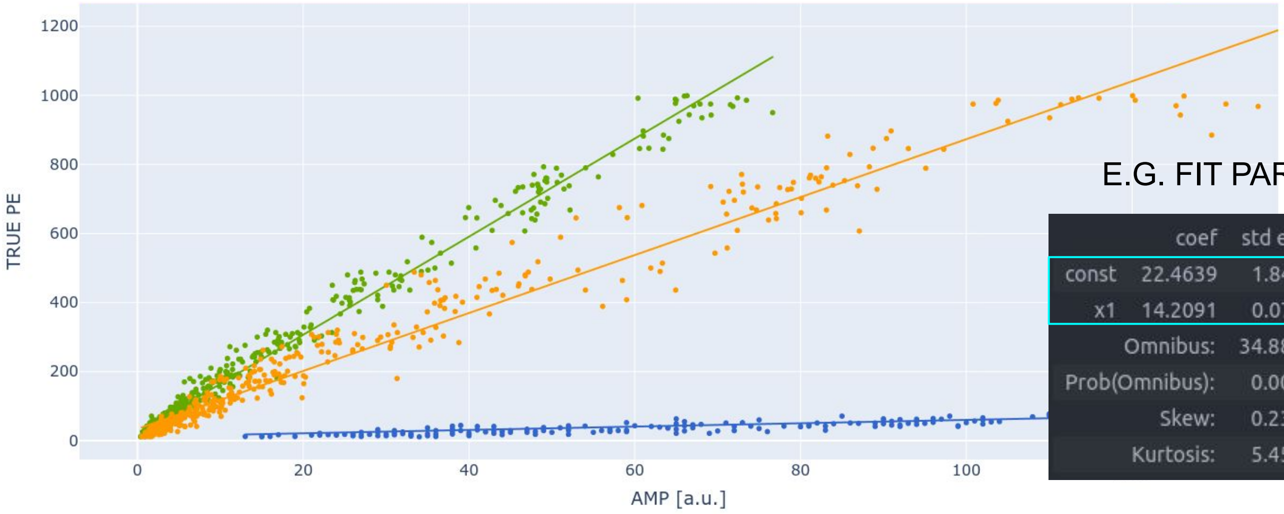
ANALYSIS FRAMEWORK

- Currently we are using **3GeV muon** sample. We aim to produce dec. wvfs with all previously mentioned filter configurations and analyze resulting: **Amplitude linearity**, **charge and t0 recovery**.
- Additionally provided a analysis module to be used in larsoft.
- Standalone interactive python notebook that read the output of larsoft's analysers (digitizer, deconvolution).



AMP COMPARISON AND CALIBRATION

- A plot of amp vs true PE can be used to obtain a scaling value that would bring the deco wvfs to the original amplitude of the raw signal (-> avoid changing thresholds in other modules).
- Of course SPE area would still need to be updated (even if abs amplitude is corrected)

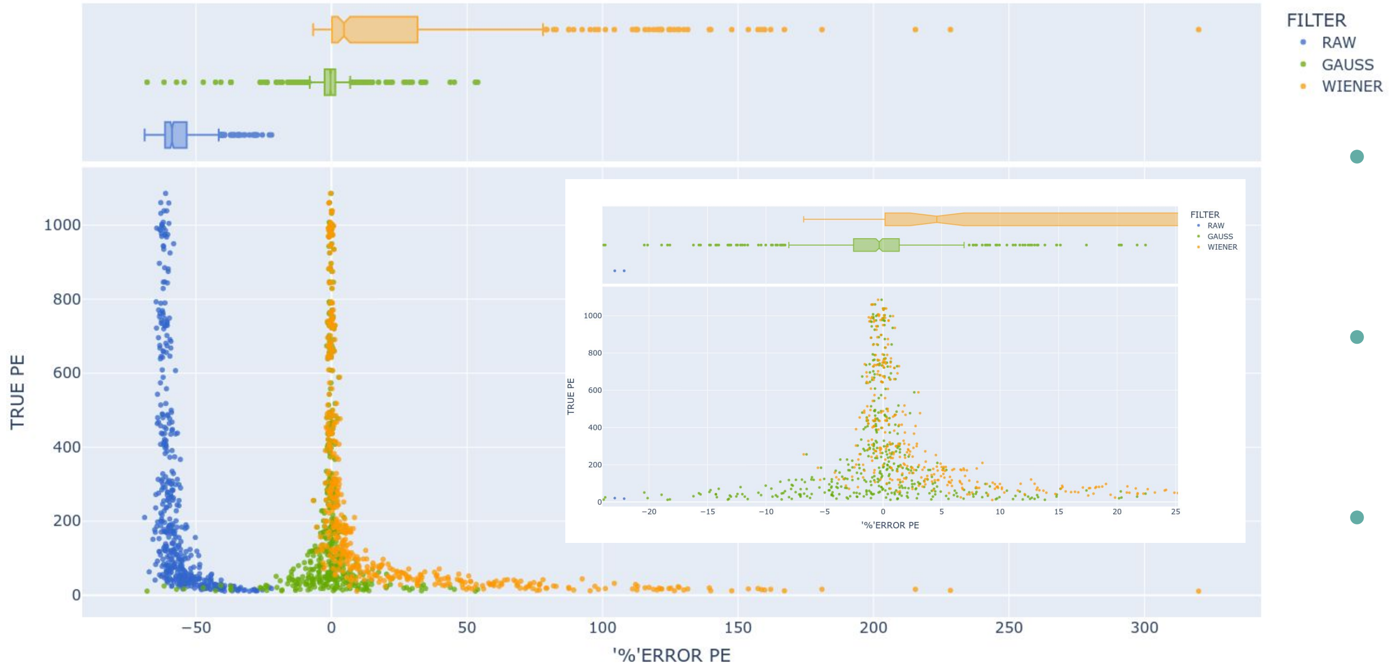


E.G. FIT PARAMETERS FOR GAUSS:

	coef	std err	t	P>|t|	[0.025	0.975]
const	22.4639	1.843	12.186	0.000	18.841	26.087
x1	14.2091	0.076	186.017	0.000	14.059	14.359

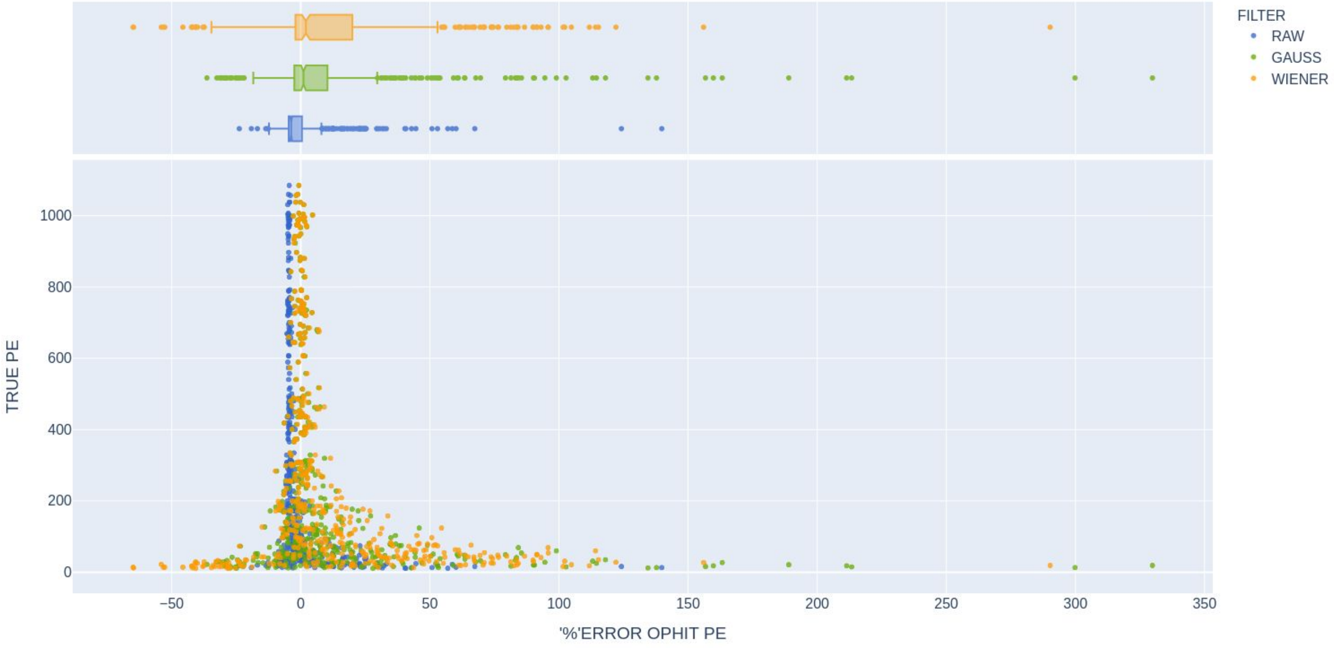
Omnibus:	34.884	Durbin-Watson:	1.476
Prob(Omnibus):	0.000	Jarque-Bera (JB):	119.214
Skew:	0.231	Prob(JB):	1.30e-26
Kurtosis:	5.451	Cond. No.	31.1

#PE RECOVERY (ESTIMATED)



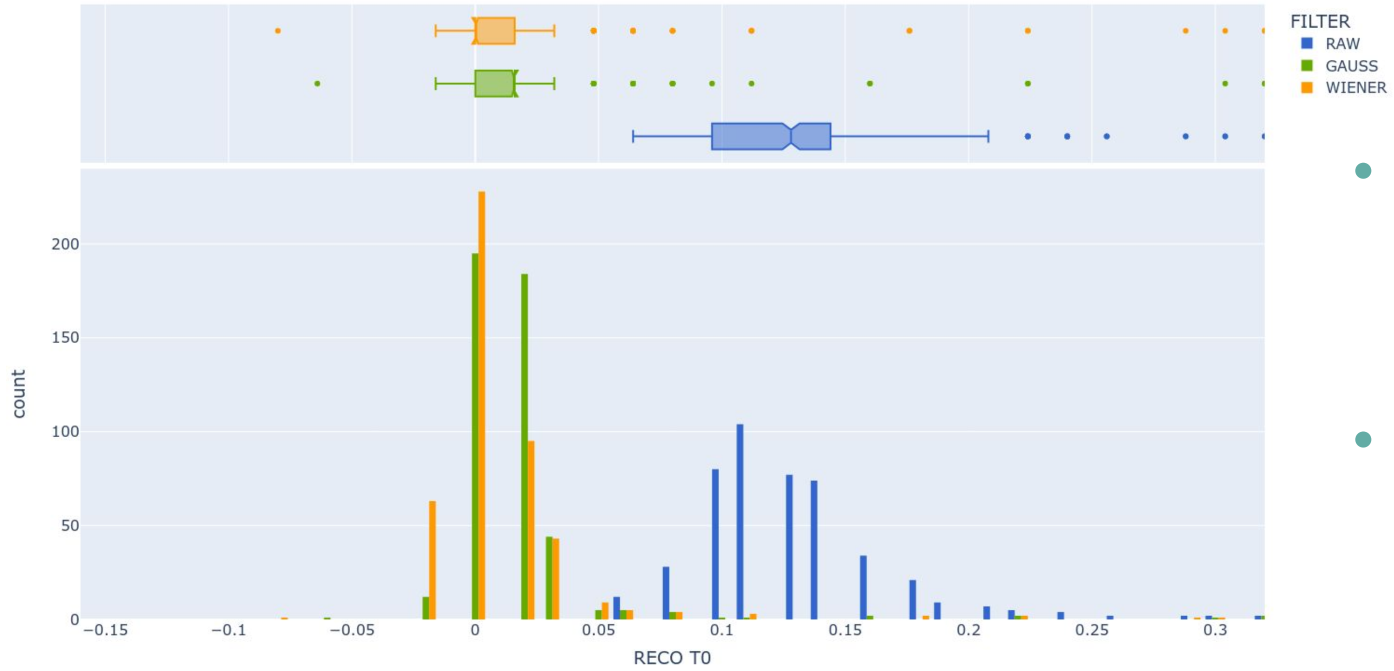
- **RAW RECO PEs** are calculated from the **positive part of wvf** and template.
- **DECO RECO PEs** are calculated **summing the whole** array.
- Very naive reco strategy. Results should improve with ophitfinder.

#PE RECOVERY (OPHITFINDER)



- **OpHitFinder Algo: SlidingWindow.**
- **RAW RECO PEs** are calculated from the calibration of a SPE wvf.
- **DECO RECO PEs.** The area of a SPE is by definition 1*scaling.

T0 RECOVERY

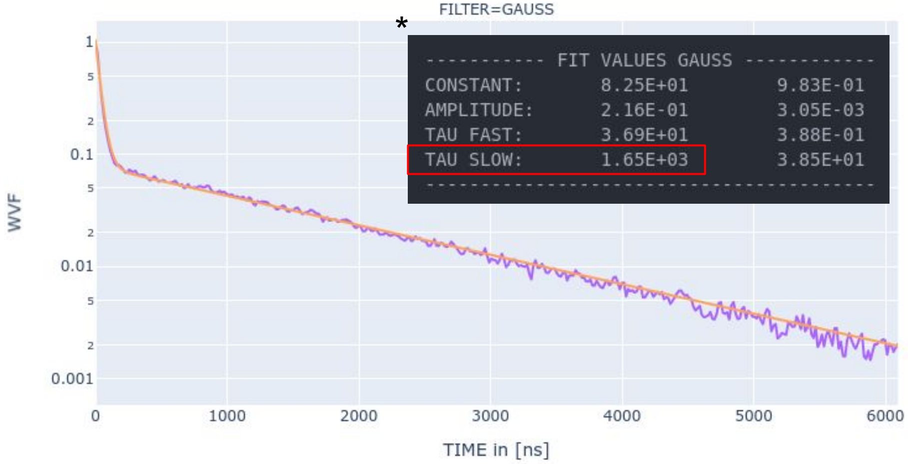


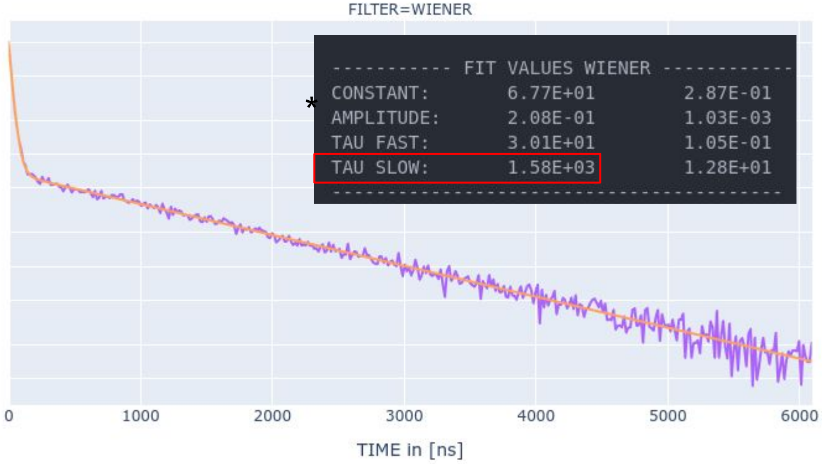
- Deconvolution can be used recover t0 with an easy calculation of the peak time.
- For a row wvf this would require a more sophisticated algorithm.

SCINTILLATION FIT

$$S(t) = \text{Const} \cdot \left[\frac{A_{\text{fast}}}{\tau_{\text{fast}}} \exp^{t/\tau_{\text{fast}}} + \frac{A_{\text{slow}}}{\tau_{\text{slow}}} \exp^{t/\tau_{\text{slow}}} \right]$$

* $A_{\text{fast}} = \text{AMPLITUDE}$
$A_{\text{slow}} = 1 - \text{AMPLITUDE}$





CURRENT PROBLEMS

- OpHitFinder algorithms work with integers:
    - Deconvolved wvfs are returned as floats.
    - **Possible workaround -> rescale recob::OpWaveform and change SPE area config.**

- New TimeStamp class:
    - raw:OpDetWaveform returns TimeStamp_t which is a double.
    - recob::OpWaveform returns a raw::RDTimeStamp which returns a ULong64_t.
        - Conversion from original TimeStamp is not possible. **Ideas?**
        - **Possible workaround -> keep using TimeStamp from raw:OpDetWaveform in OpHitFinder and other modules.**

NEXT STEPS

- OpHit finder now works with new class recob::OpWaveform (but only with scaling as a workaround).
  - Ideal algorithm settings still to be found.
  - In future we would like to write an algorithm that works natively with floats.

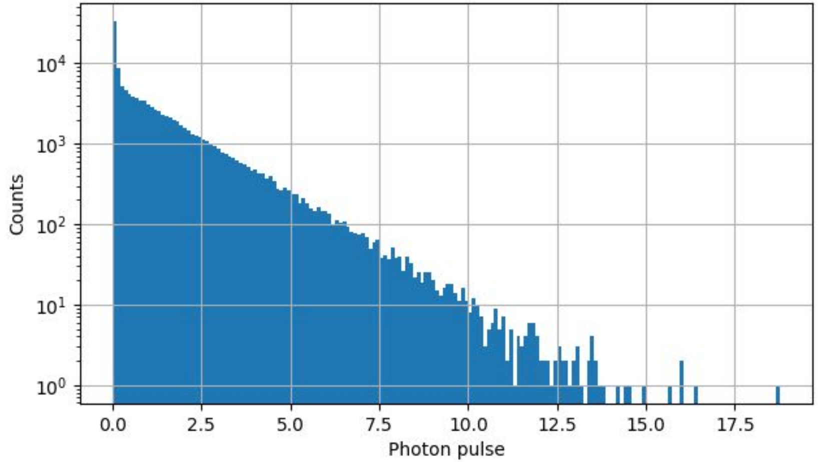
- To complete the workflow:
  - TimeStamp of the raw wvfs needs to be imported to the OpHitFinder module.
  - Provide fcl file configs for all 3 modules (digitizer, deconvolution, ophitfinder)

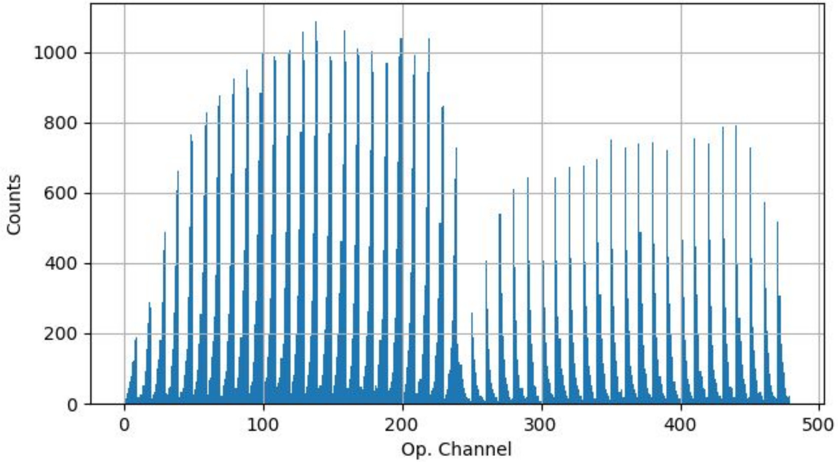
- A pull request will be made in the coming days of the OpHitFinder and the Deconvolution modules, both of which will be considered in the next HD MC production.

BACKUP

TEST SAMPLE

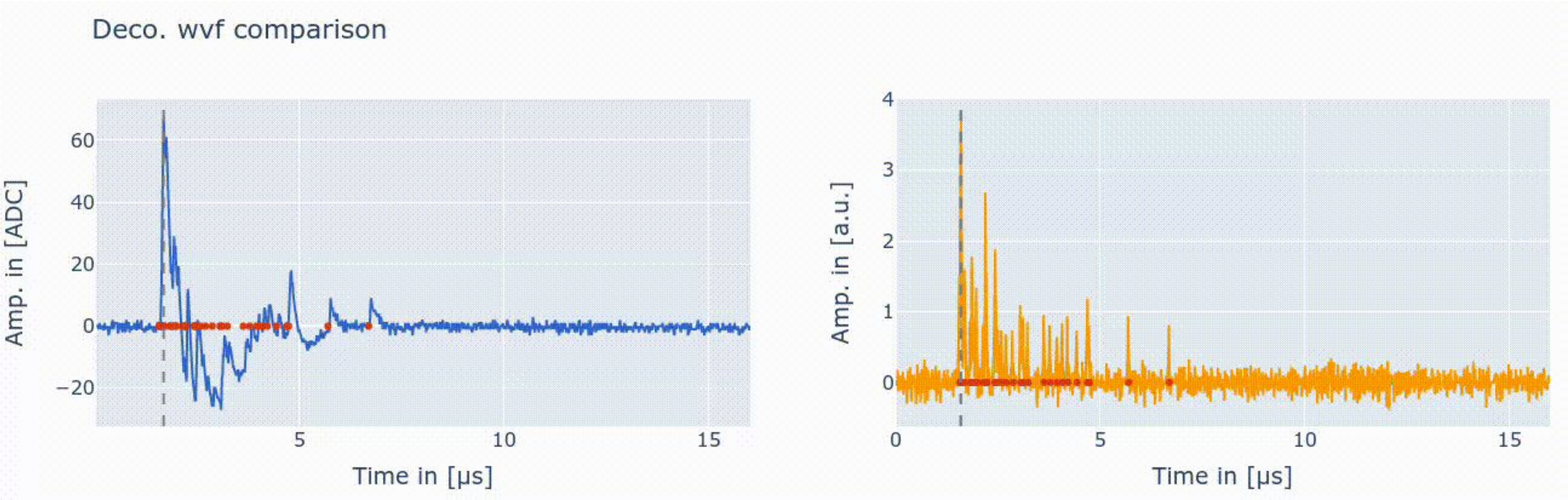
- Currently we are using **3GeV muon** sample. We aim to produce dec. wvfs with all previously mentioned filter configurations and analyze resulting:
  - **Amplitude linearity**, **charge and t0 recovery**.



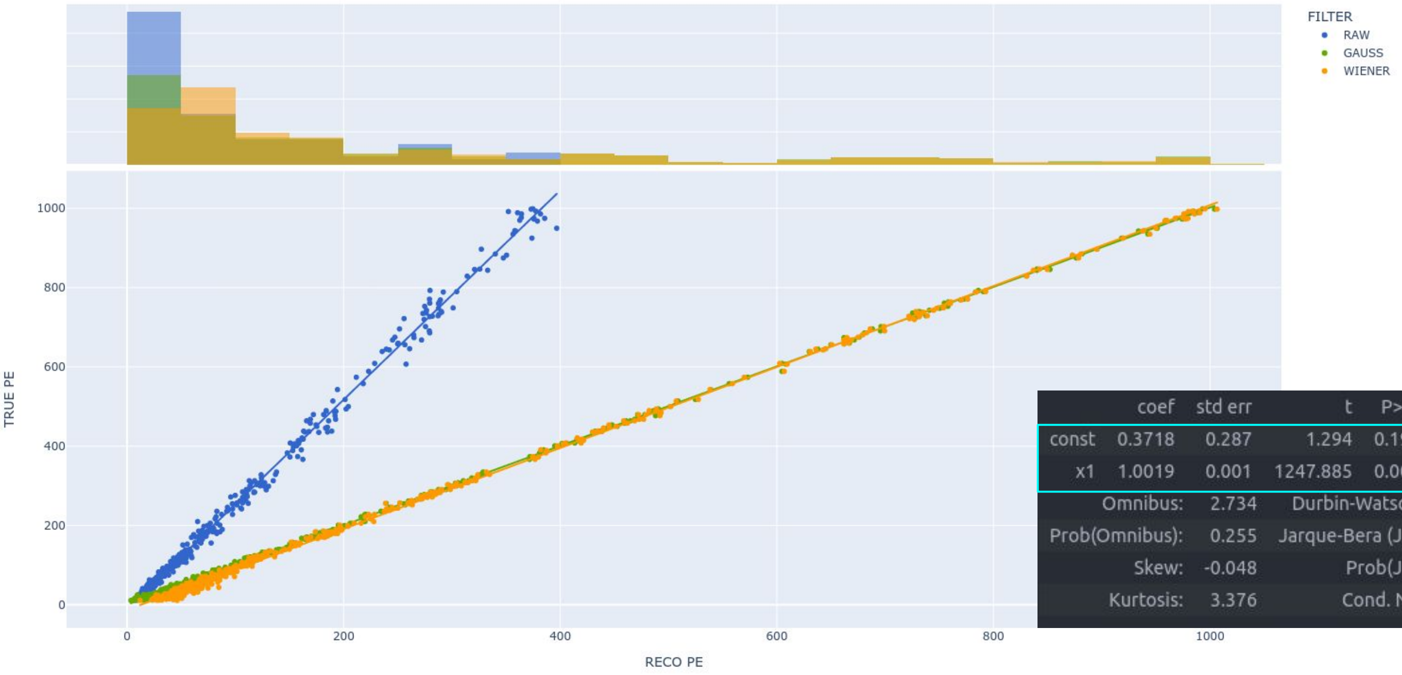


ANALYSIS FRAMEWORK

- Additionally provided a analysis module to be used in larsoft.
- Standalone interactive python notebook that read the output of larsoft's analysers (digitizer, deconvolution).



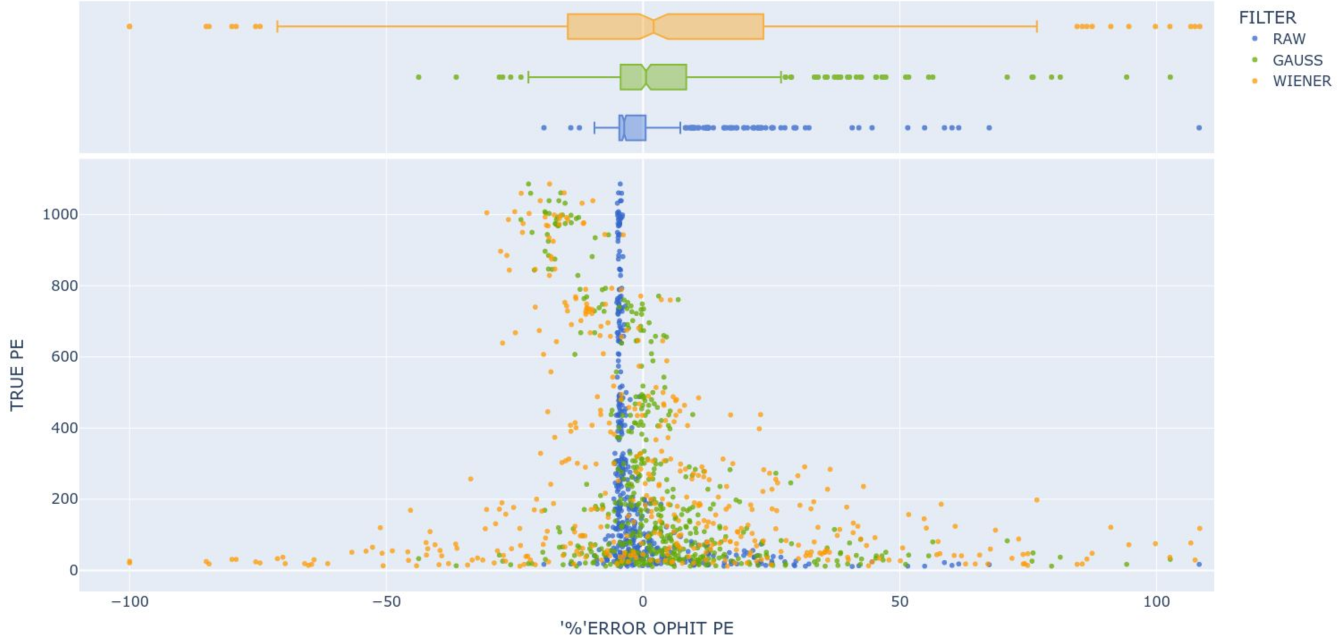
#PE RECOVERY



	coef	std err	t	P>|t|	[0.025	0.975]
const	0.3718	0.287	1.294	0.196	-0.193	0.936
x1	1.0019	0.001	1247.885	0.000	1.000	1.003

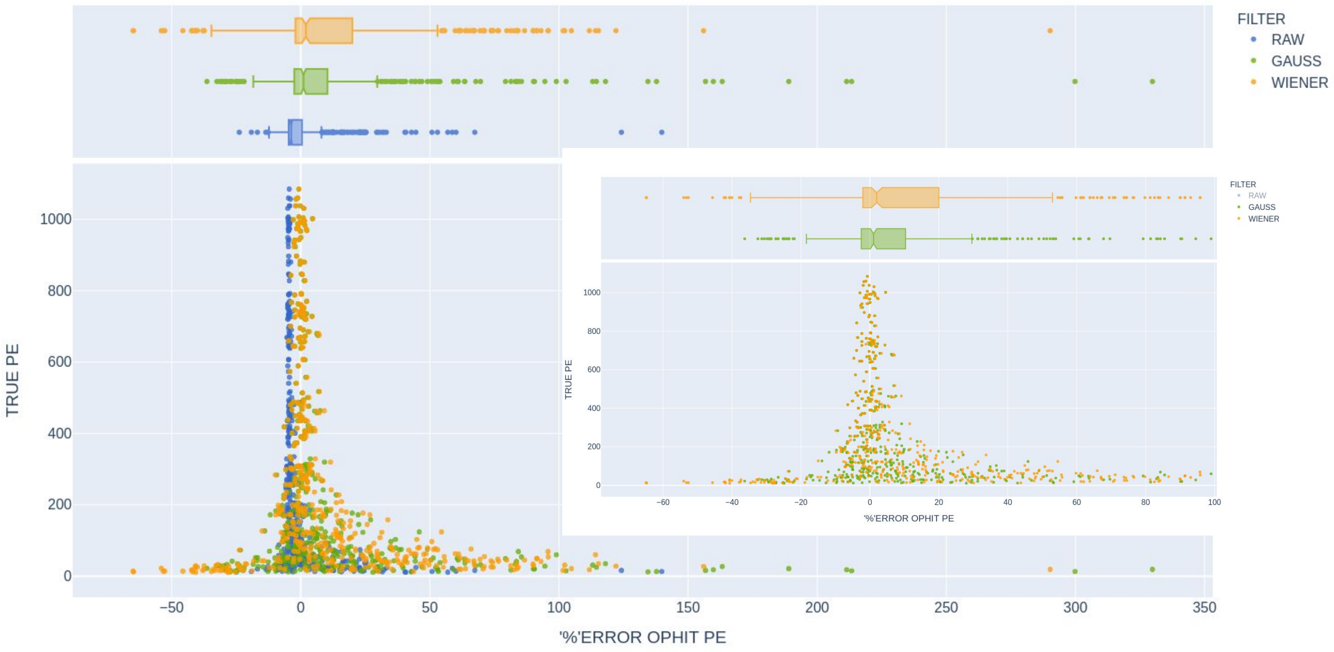
Omnibus:	2.734	Durbin-Watson:	2.078
Prob(Omnibus):	0.255	Jarque-Bera (JB):	2.888
Skew:	-0.048	Prob(JB):	0.236
Kurtosis:	3.376	Cond. No.	479.

#PE RECOVERY (OPHITFINDER) WITHOUT POSTFILTER



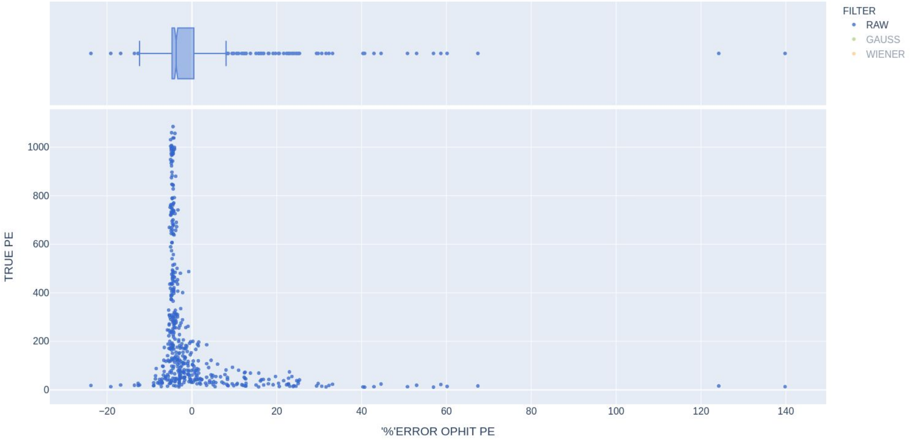
- **OpHitFinder Algo. SlidingWindow**
- **RAW RECO PEs** are calculated from the **ideal wvf**.
- **DECO RECO PEs**.

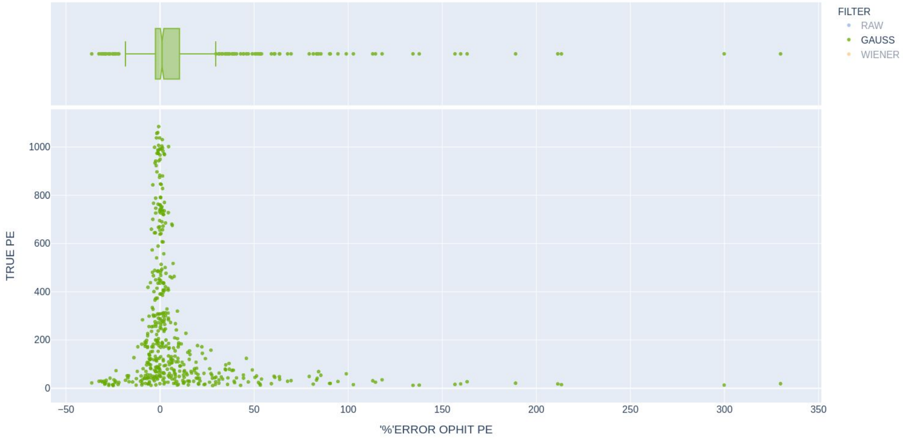
#PE RECOVERY (OPHITFINDER) WITH POSTFILTER



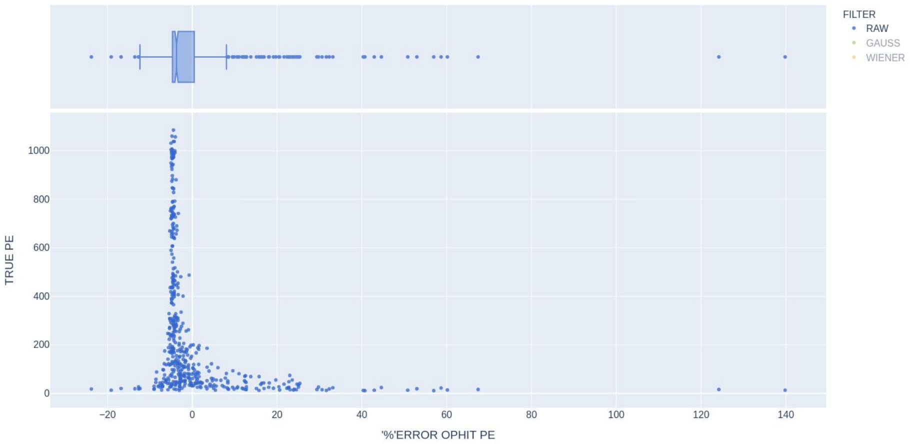
- **OpHitFinder Algo: SlidingWindow.**
- **RAW RECO PEs** are calculated from the calibration of a SPE wvf.
- **DECO RECO PEs**. The area of a SPE is by definition 1*scaling.

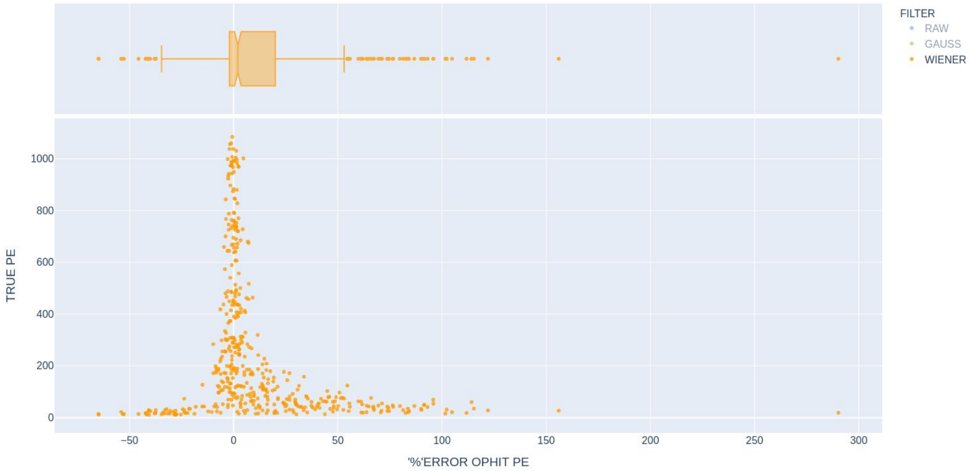
#PE RECOVERY (OPHITFINDER) WITH POSTFILTER



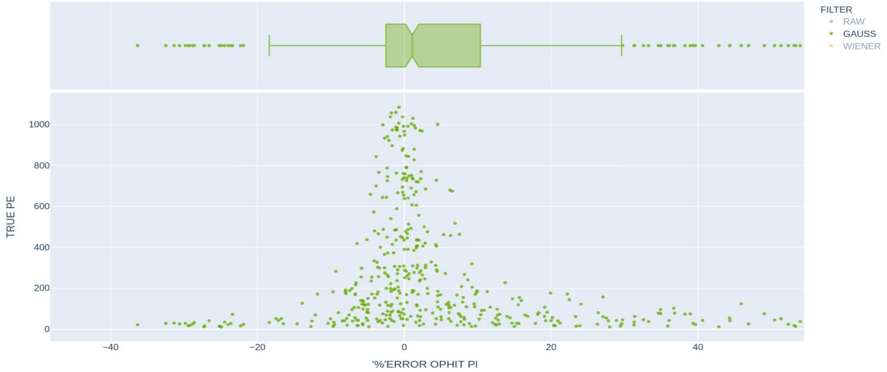


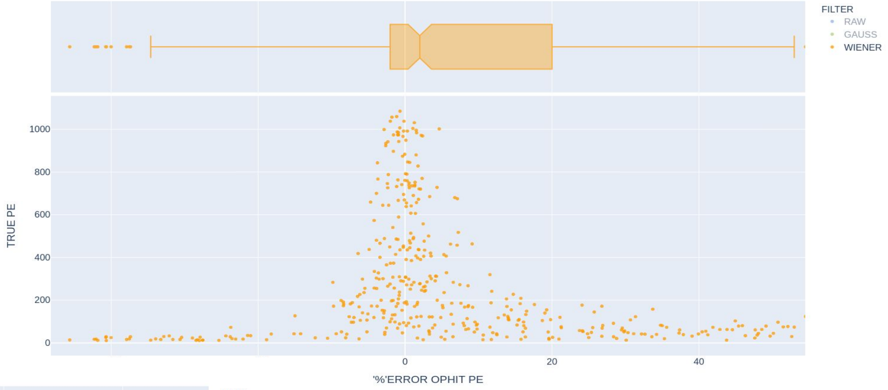
#PE RECOVERY (OPHITFINDER) WITH POSTFILTER

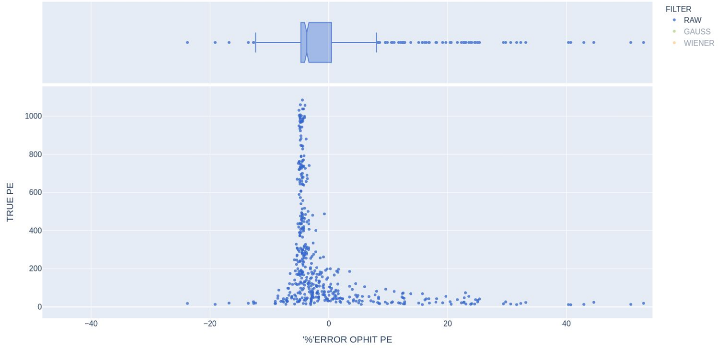




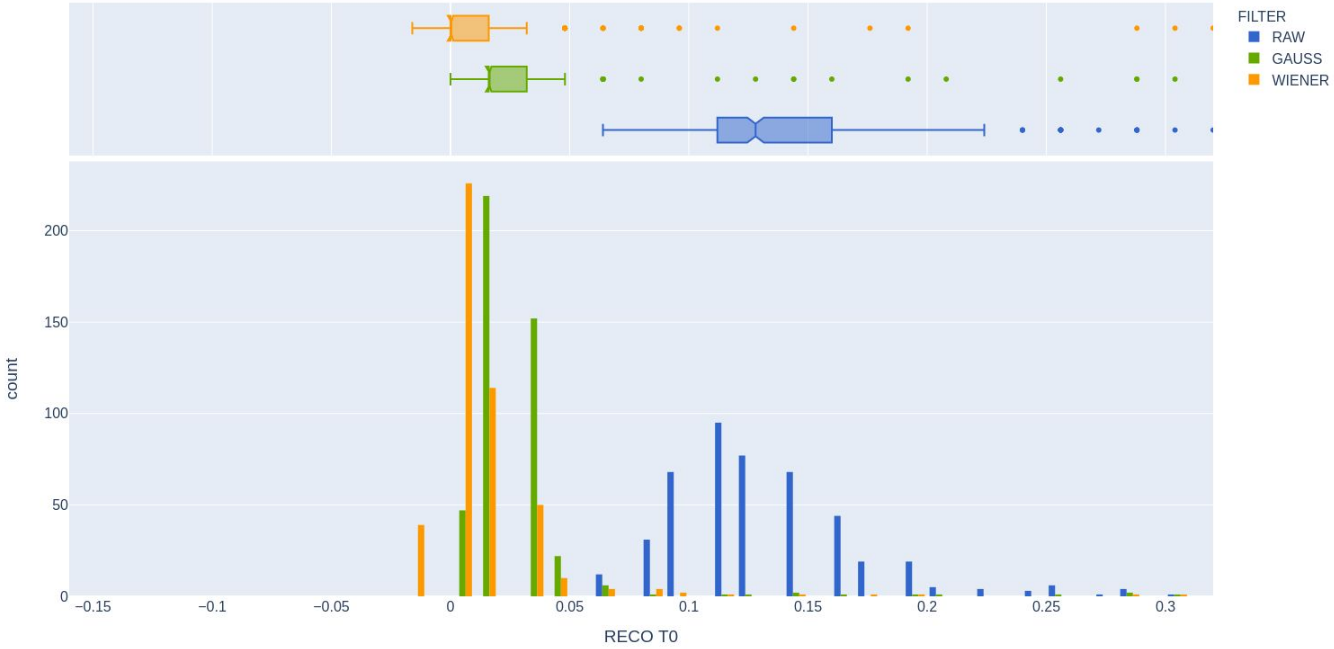
#PE RECOVERY (OPHITFINDER) WITH POSTFILTER







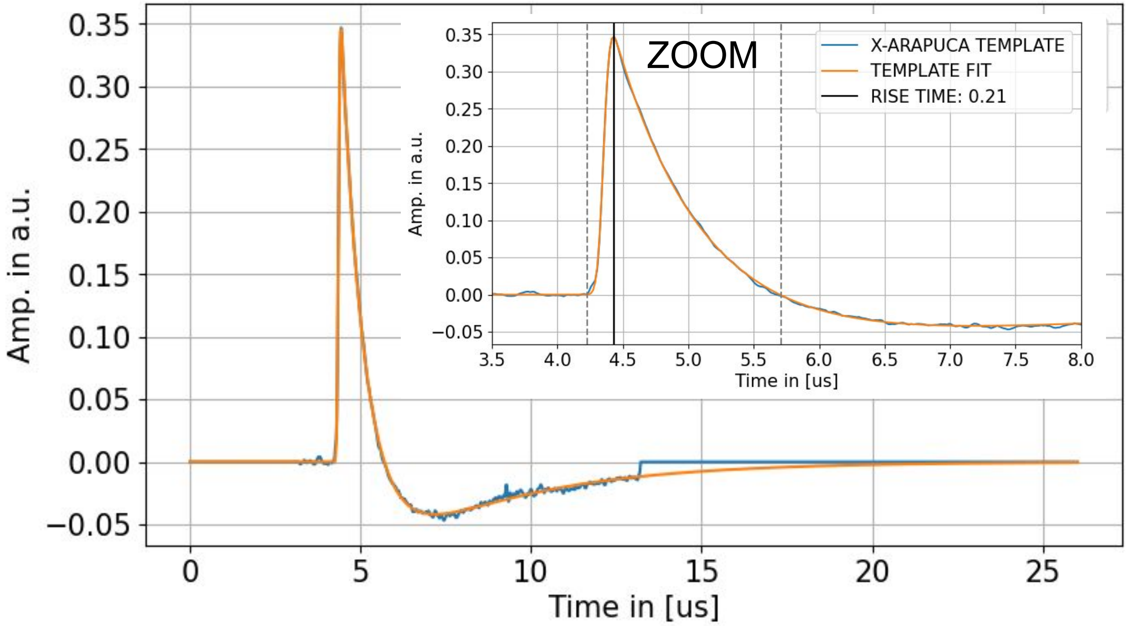
T0 RECOVERY WITH POSTFILTER



- Deconvolution can be used recover t0 with an easy calculation of the peak time.
- For a row wvf this would require a more sophisticated algorithm.

X-ARAPUCA SPE

Deconvolution algorithm tested on one of the latest available wvf template*.



The rise time **(0.21us)** of the signal measured from baseline to max amplitude.

Orange wvf is a **fit** (details in backup).
-> Use to remove the residual noise of the template.

Deconvolution Method → Analyse frequency space → Check consistency with SPE and MC simulation

Theoretical **deconvolution method:**

- Use ideal SPE signal **as deconvolution template**.
- Calculate **Wiener** filter from signal (S) and noise model (N).
- **Adjust Gauss curve to filter correspondent frequencies**.
- Divide filtered signal in frequency space.
- Compute baseline of deconvolved wvfs.