D2O Analysis Pipeline Documentation

Gen Li
Carnegie Mellon University

January 6, 2026

Abstract

This report documents the Python-based analysis pipeline developed for the D2O (Heavy Water) Detector. The pipeline is designed to process raw ROOT files containing PMT and SiPM waveform data. The system features automatic multi-Gaussian calibration for low-light spectra, vectorized event selection, and memory-efficient incremental aggregation. This document serves as both a technical reference for the codebase and a user guide for configuration and execution.

Contents

1 Pipeline Overview

1.1 Introduction

The D2O analysis pipeline processes raw data from the detector to extract key physical quantities including Photoelectrons (P.E.), time differences (Δt), and Multiplicity. The workflow is split into two primary stages:

1. **Parallel Processing (Map Phase):** The run range is split into sub-jobs. Each job processes individual ROOT files to apply calibrations, cuts, and generate intermediate binary data.
2. **Aggregation (Reduce Phase):** Processed outputs are combined to generate global statistics, fits, and final plots.

2 Script Introduction

2.1 Core Configuration

2.1.1 `config.py`

Role: Single Source of Truth for analysis parameters and toggles.

What it controls (high level).

- **Input locations and file suffixes:** `DATA_DIR_M1`, `DATA_DIR_M2`, and per-phase ROOT suffixes.
- **Event selection and timing cuts:** pile-up window (`TIME_INTERVAL_CUT_NS`), Δt window (`DELTA_T_CUT`), P.E. range (`PE_CUT`), time-spread cut (`TIME_STD_CUT`), multiplicity definition and threshold (`MULTIPLICITY_SPE`, `MULTIPLICITY_CUT`).
- **Histogram/plot settings:** binning for P.E. and veto efficiency (`BINS`, `VETO_BINS`, `VETO_RANGE`), log-scale flags.
- **Fit settings:** low-light SPE fit range (`LOW_LIGHT_FIT_RANGE`), exponential lifetime fit settings (`DO_TAU_FIT`, `TAU_FIT_WINDOW`).
- **Optional analysis modules:** Thin-veto and BRN analysis toggles and their histogram configs.

2.2 Job Orchestration

2.2.1 `submit_veto.sh`

Role: End-to-end driver: submits the parallel processing jobs *and* schedules the master aggregation as a dependent job.

What it does.

1. **Defines run range and phase:** `start_run`, `end_run`, `M1_or_M2`, and number of sub-jobs `njobs`.

2. **Creates a unique top-level output directory**: `analysis_<start>-<end>_<M1/M2>_<timestamp>/`.
3. **Snapshots the code:** copies all `*.py` into `analysis_.../code/` and runs from that snapshot (reproducibility).
4. **Splits the run range into chunks** and submits one SLURM job per chunk, each calling:

```
python Read_Cut_Hist_D2O_multi_veto.py <job_start> <job_end> <M1_or_M2>
    <TOP_OUTPUT_DIR>
```

5. **Submits the master aggregation job** (`aggregate_master_veto.py`) with `afterok` dependency on *all* processing job IDs, so it runs only after all sub-jobs complete successfully.

2.2.2 `submit_aggregation_only.sh`

Role: Utility script to re-run only the master aggregation on an *existing* analysis directory.

When to use it.

- Aggregation failed (e.g., memory) but the `subjob_*` outputs exist.
- You changed only **plotting / fit / binning** settings in `config.py` (or in the aggregation code) and want to regenerate `MASTER_RESULTS` without reprocessing all runs.

2.2.3 Expected Output (What you should see on disk)

A. Top-level batch directory (created by `submit_veto.sh`).

- `analysis_<start>-<end>_<M1/M2>_<timestamp>/`
- `analysis_.../code/` (snapshot of python sources for reproducibility)
- One directory per sub-job chunk: `subjob_<job_start>-<job_end>/`
- `MASTER_RESULTS/` (created by master aggregation)

B. Per-run debug products (inside each `subjob_*` directory). For each run you will see:

- `run<run>_<timestamp>/time_length.json` (livetime for that run)
- `run<run>_<timestamp>/histograms/`
  - `<label>_<M1/M2>_correlation_map.png`
  - (optional) thin-veto plots, if enabled: `thin_veto_height.png`, `thin_veto_area.png`
  - (optional) BRN plots, if enabled: `brn_delta_t.png`, `brn_area.png`
- `run<run>_<timestamp>/cuthist/`
  - `delta_t_hist_<M1/M2>.png` and `delta_t_hist_<M1/M2>.pkl`
  - `total_pe_hist_<M1/M2>.png` and `total_pe_hist_<M1/M2>.pkl`
- `run<run>_<timestamp>/lowlight/`
  - `<label>_<M1/M2>_low_light_fit.png` and `<label>_<M1/M2>_low_light_fit.pkl`

C. Per-subjob (chunk) aggregation products (consumed by master). At the end of each sub-job, the worker writes:

- `aggregated_delta_t.npy`, `aggregated_total_pe.npy`, `aggregated_multiplicity.npy`
- `aggregated_sipm_area_array.pkl`
- `aggregated_pe_trig2.pkl` and `aggregated_pe_trig2_or_34.pkl` (veto-efficiency inputs)
- `aggregated_low_light_hists.pkl`
- `aggregated_thin_veto_hists.pkl` (if thin-veto enabled)
- `aggregated_brn_channel_data.pkl` (if BRN enabled)
- `subjob_time_length.json` (livetime summary for the sub-job)

D. Final master outputs (inside `MASTER_RESULTS/`). Master aggregation creates:

- `aggregated_delta_t_<M1/M2>.png`
- `aggregated_total_pe_<M1/M2>.png`
- `total_time_length.json` (global livetime)
- `run_info.txt` (summary of what was found/used)
- Veto performance summary: `<label>_<M1/M2>_total_pe_comparison_master.png`, `<label>_<M1/M2>_veto_efficiency_master.png` and `.pkl`
- Low-light global SPE fits: `<label>_<M1/M2>_low_light_fits.png` and `.pkl`
- SiPM area summary: `<label>_<M1/M2>_sipm_area_histograms.png` and `.pkl`
- (optional) BRN master plots: `<label>_<M1/M2>_brn_delta_t_master.png/.pkl`, `<label>_<M1/M2>_brn_area_master.png/.pkl`

2.3 Analysis Logic (How everything works together)

2.3.1 `Read_Cut_Hist_D2O_multi_veto.py` — Worker / Map step

Role: Process one run at a time, generate per-run QA plots, and emit per-subjob aggregated arrays/pickles for the master reducer.

Entry point: `main()`. A worker job is invoked as:

```
python Read_Cut_Hist_D2O_multi_veto.py <start_run> <end_run> <M1_or_M2> <
    top_output_dir>
```

It creates `subjob_<start>-<end>/` under `top_output_dir` and loops over runs in that range.

Run-level processing flow (`RunProcessor.process_run`). For each run, the worker:

1. **Creates a timestamped run directory** (with `histograms/`, `cuthist/`, `lowlight/`).
2. **Loads event-level arrays** from the processed ROOT file and constructs a working event table.
3. **Computes derived quantities** using vectorized helpers (total P.E., multiplicity, time spread, Δt).
4. **Applies event-quality and physics cuts** (pile-up rejection, Δt window, P.E. cuts, multiplicity/time-spread cuts).
5. **Writes per-run QA artifacts**:
   - Cut-flow histograms to `cuthist/` (`delta_t_hist_*`, `total_pe_hist_*`).
   - Correlation heatmap(s) to `histograms/`.
   - Low-light SPE fits to `lowlight/`.
   - Optional thin-veto and BRN plots to `histograms/` if enabled in `config.py`.
6. **Returns run outputs** back to `main()` so the sub-job can append them to the chunk-level aggregates.

Major components and functions (what they do).

- `RunProcessor.process_run(...)`: top-level per-run driver (directory creation, calling the processing sub-steps, returning results + livetime).
- `RunProcessor._get_run_time_length(...)`: measures and writes `time_length.json` for the run.
- `DataProcessor.calculate_total_pe(...)`: computes event-wise total P.E. from per-channel P.E.
- `DataProcessor.compute_delta_t(...)`: computes correlated Δt between a muon trigger and candidates for correlated-event studies.
- `LowLightAnalyzer.fit_and_plot_low_light(...)`: per-run constrained multi-Gaussian SPE modeling on low-light events; writes fit plots and serialized fit results.
- `Plotter._save_cut_histograms(...)`: writes the `cuthist/` products (both `.png` and `.pkl` with histogram content).
- `Plotter.plot_correlation_maps(...)`: correlation heatmap saved as `*_correlation_map.png`.
- `ThinVetoAnalyzer.process_thin_veto(...)` (optional): builds thin-veto histograms and per-run plots.
- `BRNAnalyzer.process_brn(...)` (optional): builds BRN time/area distributions and per-run plots.

Subjob-level outputs (handoff to Reduce). After looping over runs, `main()` concatenates/accumulates results and writes the per-subjob `aggregated_*.npy/.pkl` files plus `subjob_time_length.json`. These are the only inputs the master reducer `needs` to produce global plots.

2.3.2 `aggregate_master_veto.py` — Master / Reduce step

Role: Combine all `subjob_*` outputs into global arrays/histograms, produce final publication-ready plots, and write a run summary.

Master flow (`MasterAggregator.run`).

1. Creates `MASTER_RESULTS/` under the top analysis directory.
2. Scans for `subjob_*` directories and loads:
   - main arrays (`aggregated_delta_t.npy`, `aggregated_total_pe.npy`, `aggregated_multiplicity.npy`)
   - veto-efficiency inputs (`aggregated_pe_trig2*.pkl`)
   - low-light binned hist data (`aggregated_low_light_hists.pkl`)
   - SiPM binned data (`aggregated_sipm_area_array.pkl`)
   - optional thin-veto and BRN products (if present)
   - livetime accounting from `subjob_time_length.json`
3. Produces final global plots:
   - global Δt and total P.E. (`aggregated_delta_t_*.png`, `aggregated_total_pe_*.png`)
   - veto efficiency + trigger-comparison plots
   - global low-light SPE fits and SiPM area histograms
   - optional BRN master plots
4. Writes bookkeeping outputs: `total_time_length.json` and `run_info.txt`.

3 Configuration & Setup Guide

3.1 Prerequisites

- A SLURM environment.
- Python environment with the analysis dependencies used by the scripts (numpy/pandas/matplotlib/scipy/uproot, etc.).
  I am using default Python 3 on the `ernest` machine, which is `/usr/bin/python3`.
- Access to the processed ROOT files for the chosen phase (M1 or M2).

3.2 Transfer Data

We recommend using Globus to transfer D_2O data from ORNL phylogin1 to MEG. A step-by-step note is available in Becca's LabArchives entry: `Transfer terabytes of data`

Globus will require your `ORNL guest portal`'s username and password to connect to ORNL Physics DTN.

As of January 5, 2026, the D_2O data at ORNL are located on phylogin1 cluster at:

`/data41/coherent/data/d2o/`

3.3 What to Edit Before Running (One-time / Per-system)

3.3.1 1) Point `config.py` to your data

Set:

```
DATA_DIR_M1 = "/path/to/M1_data"
DATA_DIR_M2 = "/path/to/M2_data"
suffix_M1   = "..."   # expected processed ROOT suffix for M1
suffix_M2   = "..."   # expected processed ROOT suffix for M2
```

3.3.2 2) Configure cuts / plotting defaults in `config.py`

Typical parameters you may tune:

- `DELTA_T_CUT`, `PE_CUT`, `TIME_STD_CUT`, `MULTIPLICITY_CUT`
- `LOGSCALE_*` and `BINS`/`VETO_BINS`/`VETO_RANGE`
- `LOW_LIGHT_FIT_RANGE`, `DO_TAU_FIT`, `TAU_FIT_WINDOW`
- Optional modules: `PERFORM_THIN_VETO_ANALYSIS`, `PERFORM_BRN_ANALYSIS`

3.3.3 3) Set run range + output base in `submit_veto.sh`

Edit:

```
SCRIPT_DIR="/path/to/your/Codes"
start_run=...
end_run=...
M1_or_M2="M1"   # or "M2"
njobs=...
DATA_BASE_DIR="/path/to/output_base"  # script selects M1/M2 default if you
    keep it
```

3.4 How to Run (Standard Workflow)

3.4.1 A) Full processing + aggregation (recommended for new run ranges)

Use `submit_veto.sh` when:

- This is a **new** run range, or
- You changed logic that affects per-run derived data (cuts, definitions of P.E./Δt/multiplicity, or optional modules), and you need to regenerate `subjob_*` outputs.

Command. From a SLURM login node:

```
sh submit_veto.sh
```

This will:

- create a new `analysis_*` directory,
- snapshot your code into `analysis_*/code/`,
- submit all worker jobs, and
- automatically submit the master aggregation with `afterok` dependency.

3.4.2 B) Re-run aggregation only (fast iteration on plots/fits)

Use `submit_aggregation_only.sh` when:

- The `subjob_*` directories already exist (processing completed), and
- You only want to regenerate `MASTER_RESULTS` (e.g., tweak binning, log scales, fit windows, aesthetics).

Steps.

1. Edit `submit_aggregation_only.sh`:
   - `SCRIPT_DIR` = where `aggregate_master_veto.py` lives
   - `ANALYSIS_DIR` = the top-level `analysis_*` folder that contains the `subjob_*` directories
   - (optional) increase `MEMORY_REQ`
2. Run:

   ```
   sh submit_aggregation_only.sh
   ```

3.5 Manual Runs (Debug / Development)

If you want to run a small chunk interactively (or in a single SLURM job):

- Worker for a chunk:

```
python Read_Cut_Hist_D2O_multi_veto.py <start_run> <end_run> <M1_or_M2>
    <top_output_dir>
```

- Master aggregation:

```
python aggregate_master_veto.py <analysis_top_dir>
```

Here we attached the tech note about data structure from Tulasi, noticing the path of processed data is obselete.

A D_2O Processed Data Variable Description (from Tulasi Subedi)

D2O Processed Data Variable Description

Currently, the processed data is stored at */data13/coherent/data/d2o/processedData* in *phylogin1* cluster which can be accessed by ssh with the command:
"ssh your_username@phylogin1.phy.ornl.gov".
data13 is the current disk, and once it is full, the location will change to a different disk number. The processed data is in root format with filenames structured as run####_processed_v5.root (eg. run18735_processed_v5.root). #### is the run number and v5 is the current version of the processed data. The document corresponds to version v5. For changes in older versions, please refer to the elog.

Following are the root branches in which the data is stored.

1. **Int_t eventID**
   eventID is the event identifier, starting from 1 and incrementing with each event. During post-processing, after-pulsing events are removed, so the eventID may not be continuous.
2. **Int_t nSamples[23]**
   nSamples represents the number of samples in each waveform. Currently, it is set to 45 for all channels. There DAQ stores data from 23 channels, which includes 12 PMTs, 10 SiPM, and 1 SNS beam-on (proton-on-target) signal, also called Event61). The PMT channels are indexed from 0-11, SiPMs from 12-21, and Event61 is channel 22.
3. **Short_t adcVal[23][45]**
   adcVal contains ADC values for each channel. This is a two-dimensional array (channel number * nSamples). For example, the 45-sample waveform of first channel is stored at adcVal[0][0] to adcVal[0][44], and similarly for other channels.
4. **Double_t baselineMean[23]**
   baselineMan is the mean ADC value of first 20 samples of waveform for each channel.
5. **Double_t baselineRMS[23]**
   baselineRMS is the RMS of the ADC values of the first 20 samples in the waveform for each channel.
6. **Double_t pulseH[23]**
   pulseH is the highest ADC value in the waveform above the baseline for each channel.
7. **Double_t area[23]**
   area is the integral of the ADC values above baseline from bin 23 to 40 for each channel.
8. **Int_t peakPosition[23]**
   peakPosition is the bin number of the highest ADC value in each waveform. This can range from 20 to 44, depending on the pulse position in the waveform. The peak is typically around bin 30, but there is a jitter of about 6 bins due to CAEN electronics.
9. **Long64_t nsTime**
   nsTime represents the event time (in nanoseconds) from the start of the run. The event time corresponds to when the ADC sample of any waveforms in the event exceeds the trigger threshold. It is not the time of the first sample in the waveform.

10. Int_t triggerBits

triggerBits contains information about different trigger types. It is an integer variable where each bit corresponds to a separate trigger type. External triggers (e.g., Low-Light, High-Light, and Min-Bias) are sent by the calibration system, while PMTs, SiPMs, and Event61 has internal threshold triggers.

The table below shows the trigger types, their corresponding bit, and the integer value related as (2^{bit} = triggerBits for a single trigger type)

Trigger type	bit	Integer value
Event 61	0	1
PMT	1	2
Min-Bias	2	4
High Light	3	8
Low light	4	16
SiPM	5	32

A single event can have multiple trigger types. For example, If an event has both PMT and SiPM triggers, its triggerBits value will be 100010 (binary) which is 34 (decimal).