«Nuclei» wagon

Update

V. Kireyeu 03.06.2025

Outline

- Introduction
    - How it was
    - Some new ideas
- Implementation
- Documentation and code organisation
- Utilisation
- Summary

Introduction

How it was

Do first, think later

- JSON configuration file for the wagon 
- Documented C++ «wagon» code 
- Single Python post processing program: 
  - Many subroutines in one place 
  - Fitting procedure parallelisation: hardcoded ranges, initial parameters 
  - Wagon configuration file used for the analysis 

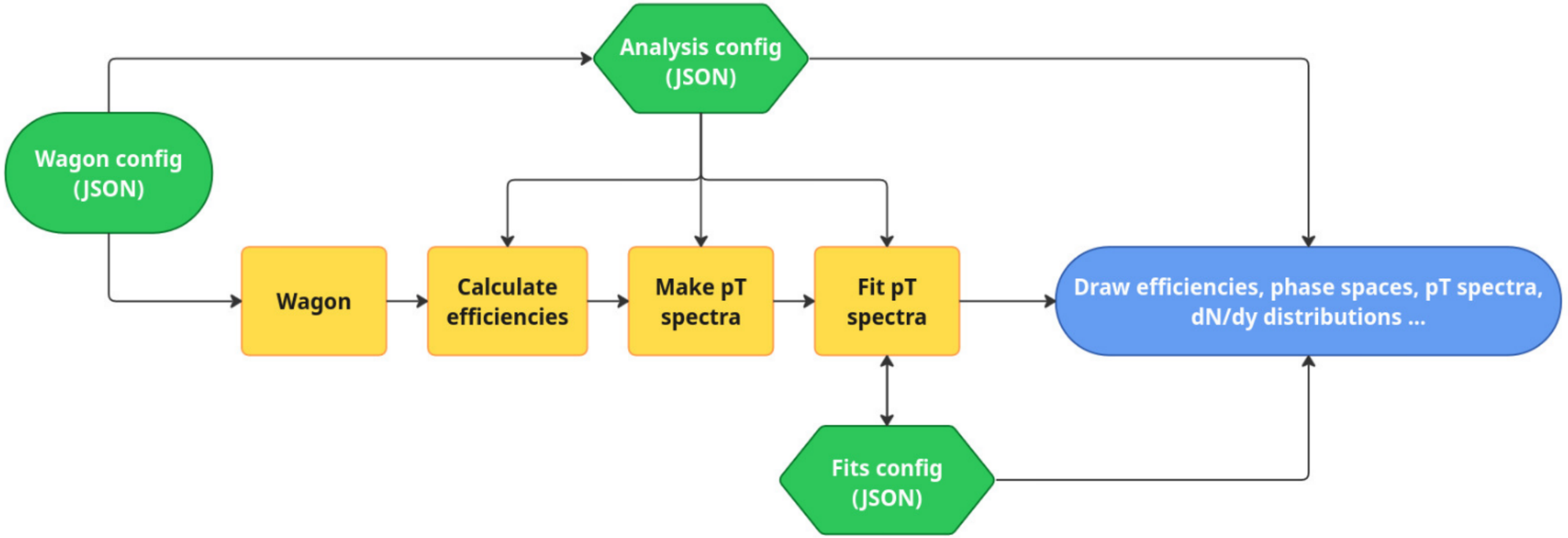
Some new ideas

Think first, do later

- Config driven analysis — move as much as possible out of the code
- Rethink the architecture, but keep old good ideas
- Old code refactoring
- New approach for the code storage, integration and deployment
- Documentation: out of the code, extended

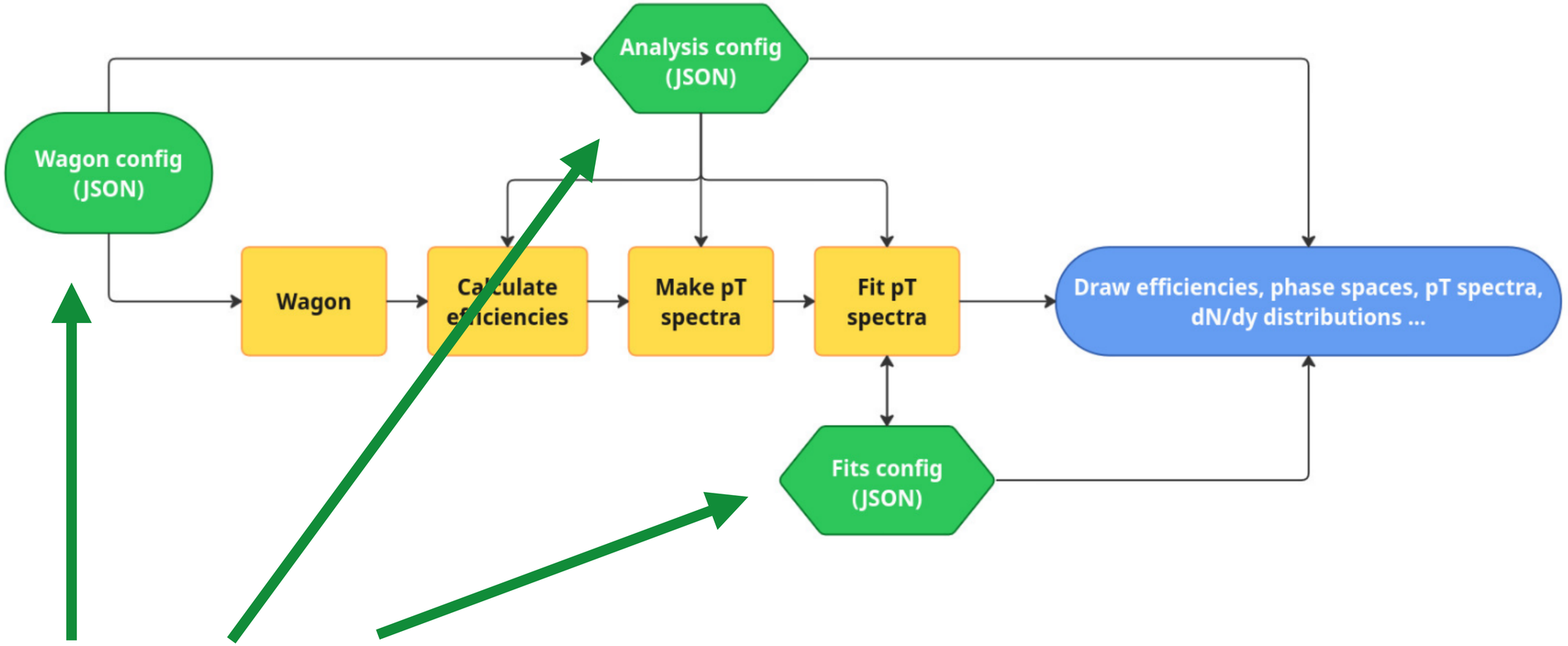
Some new ideas

How it should be



Some new ideas

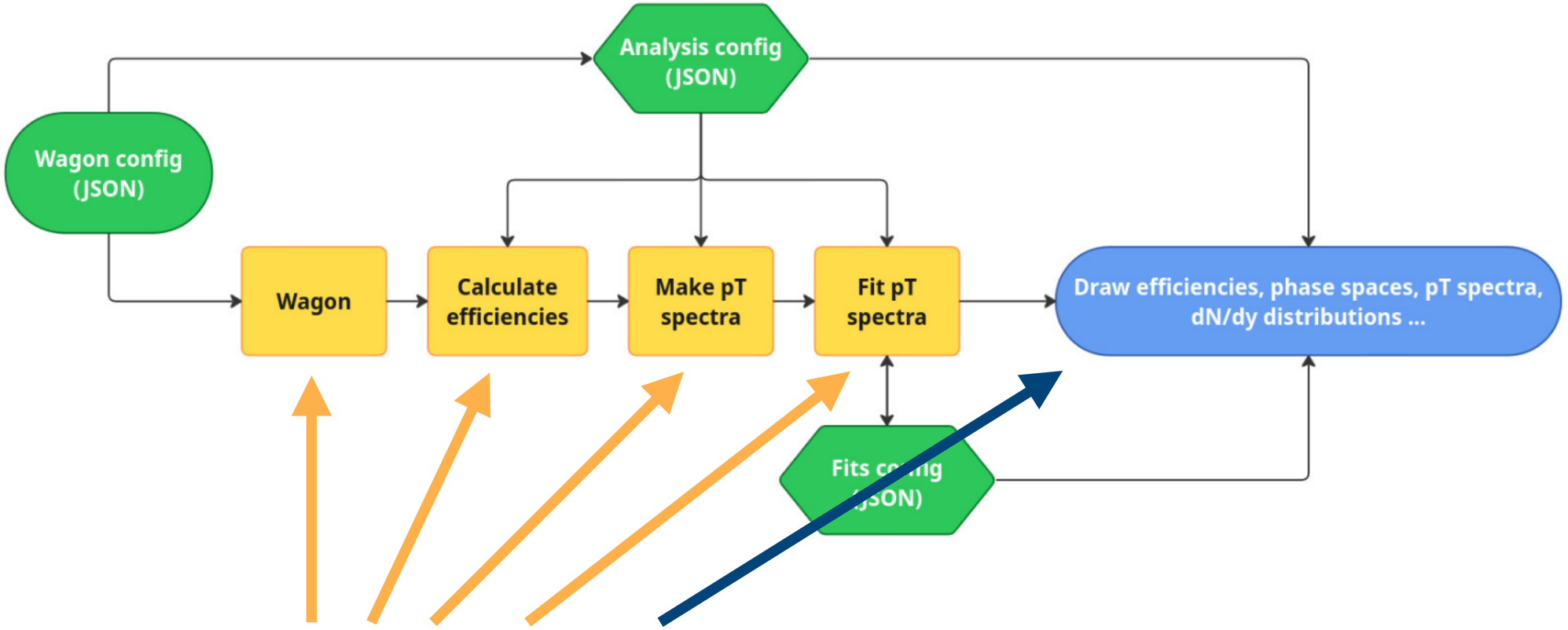
How it should be



These parts are out of the code, configurable and could/must be modified

Some new ideas

How it should be



These parts are the code and must **NEVER** be modified

Typical errors in MpdRoot wagons

which could be easily solved by the configuration files

- Hardcoded values such as particles PDG codes or PID IDs
- Hardcoded constants such as particle masses or even Pi
- Hardcoded centrality classes
- Hardcoded histograms and their bins, limits etc
- Hardcoded parameters and «magic numbers» without explanation

Typical errors in MpdRoot wagons

which could not be solved by the configuration files
but must be solved somehow

- Missing comments: purpose of the functions? Parameters? Return values?
- Lack of the documentation: «black box» type analysis — how to run it? How to use it?
- Obsolete commented-out code. Why it is here? Is it needed?
- Code duplication
- No post processing code: how to make distributions? How to get (pre-) final plots?

Implementation

trying again to avoid common mistakes

Implementation

of the new ideas

- JSON configuration file for the wagon

Left unchanged — it was a brilliant idea

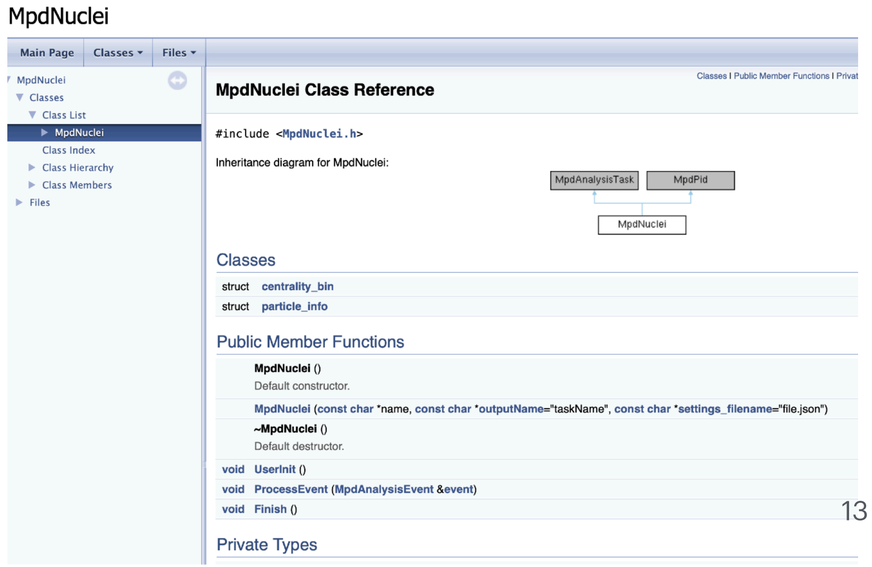

```
{
    "Verbose": "1",
    "N_MPD_PID_Particles": "8",
    "make_MC": "1",
    "make_Efficiency": "1",
    "PID_mode": "2",
    "DCA_mode": "1",
    "TOF_mode": "1",
    "use_pt_corrections": "0",
    "pt_corrections_file": "pt_corrections.root",
    "Events": {
        "PrimaryVertexZ": "100",
        "Centrality": [[0, 20], [20, 40], [40, 80]]
    },
    "Tracks": {
        "NHits": "27",
        "NSigmaDCAx": "1",
        "NSigmaDCAy": "1",
        "NSigmaDCAz": "1",
        "LowPtCut": "0.05"
    },
    "PID": {
        "TPCSigma": "2",
        "TOFSigma": "2",
        "TOFDphiSigma": "3",
        "TOFDzSigma": "3"
    },
    "MpdPid": {
        "Energy": "9.2",
        "Coef": "1.0",
        "Generator": "PHQMD",
        "Tracking": "CFHM",
        "IniString": "pikaprdetrhe3he4",
```

Implementation

of the new ideas

- Documented C++ «wagon» code

Mostly unchanged — amazingly good architecture from the beginning



Implementation

of the new ideas

Changes in the wagon code

```
std::vector<std::vector<TH2F*>> hv__eff_tpc_numerator;
std::vector<std::vector<TH2F*>> hv__eff_tpc_denominator;
std::vector<std::vector<TH2F*>> hv__cont_sec_numerator;
std::vector<std::vector<TH2F*>> hv__cont_sec_denominator;
std::vector<std::vector<TH2F*>> hv__eff_pid_numerator;
std::vector<std::vector<TH2F*>> hv__eff_pid_denominator;
std::vector<std::vector<TH2F*>> hv__cont_pid_numerator;
std::vector<std::vector<TH2F*>> hv__cont_pid_denominator;
std::vector<std::vector<TH2F*>> hv__eff_tof_denominator;
std::vector<std::vector<TH2F*>> hv__eff_pid_dedx_numerator;
std::vector<std::vector<TH2F*>> hv__cont_pid_dedx_numerator;
std::vector<std::vector<TH2F*>> hv__cont_pid_dedx_denominator;
std::vector<std::vector<TH2F*>> hv__eff_pid_tpc_numerator;
std::vector<std::vector<TH2F*>> hv__cont_pid_tpc_numerator;
std::vector<std::vector<TH2F*>> hv__cont_pid_tpc_denominator;
std::vector<std::vector<TH2F*>> hv__eff_pid_tof_numerator;
std::vector<std::vector<TH2F*>> hv__cont_pid_tof_numerator;
std::vector<std::vector<TH2F*>> hv__cont_pid_tof_denominator;
```

```
/// List of efficiencies counters
enum EffCounters {
  kMC_MCPID,          ///< 0:  MC tracks, MC PID, mc y and pt
  kRC_MCPID,          ///< 1:  RC tracks, MC PID, mc y and pt
  kRC_MCPID_R,        ///< 2:  RC tracks, MC PID, reconstructed y and pt
  kRC_MCPID_PRIM,     ///< 3:  RC tracks, MC PID, Primary by MC, mc y and pt
  kRC_MCPID_SEC,      ///< 4:  RC tracks, MC PID, Secondary by MC, reconstructed y and pt
  kRC_MCPID_TOF,      ///< 5:  RC tracks, MC PID, TOF, reconstructed y and pt
  kRC_RCPID_DEDX,     ///< 6:  RC tracks, RC PID dEdx only, reconstructed y and pt
  kRC_RCPID_DEDX_T,   ///< 7:  RC tracks, RC PID dEdx only == PDG, reconstructed y and pt
  kRC_RCPID_DEDX_F,   ///< 8:  RC tracks, RC PID dEdx only != PDG, reconstructed y and pt
  kRC_RCPID_COMB,     ///< 9:  RC tracks, RC PID combined (has ToF), reconstructed y and pt
  kRC_RCPID_COMB_T,   ///< 10: RC tracks, RC PID combined == PDG, reconstructed y and pt
  kRC_RCPID_COMB_F,   ///< 11: RC tracks, RC PID combined != PDG, reconstructed y and pt
  kRC_EVPID_TPC,      ///< 12: RC tracks, evPID TPC, reconstructed y and pt
  kRC_EVPID_TPC_T,    ///< 13: RC tracks, evPID TPC == PDG, reconstructed y and pt
  kRC_EVPID_TPC_F,    ///< 14: RC tracks, evPID TPC != PDG, reconstructed y and pt
  kRC_EVPID_TOF,      ///< 15: RC tracks, evPID TOF, reconstructed y and pt
  kRC_EVPID_TOF_T,    ///< 16: RC tracks, evPID TOF == PDG, reconstructed y and pt
  kRC_EVPID_TOF_F,    ///< 17: RC tracks, evPID TOF != PDG, reconstructed y and pt
  kNumCounters        ///< 18: Total number of counters
 };
/**  @name Efficiency histograms
 */
///@{
  std::vector<std::vector<std::vector<TH2F*>>> hv__eff_counter =
        std::vector<std::vector<std::vector<TH2F*>>>(kNumCounters); ///< Efficiencies histograms (2D pT-y phase-space).
///@}
```

- New approach to the efficiencies counters: enum list, single object —> easy to understand and to operate
- New counters can be added to the enum list —> automatic creation and initialisation of histograms

Implementation

of the new ideas

- Single Python post processing program

- Several simpler single-purpose Python programs

Analysis configuration file — the main part of the analysis

Implementation

New analysis configuration file

Config driven analysis ✅

- System string, e.g. «Xe + W @ 2.5 A.GeV».
- Efficiencies (numerators, denominators — any combination of the wagon «efficiencies counters»)
- Paths to the wagon configuration file, each program output files, files with fits initial parameters.
- Analysis rapidity intervals, centrality bins, particle species to include.
- The pT ranges (for each particle in each rapidity interval in each centrality bin) to get reconstructed dN/dy point (and thus the ranges for the Blast-Wave extrapolations).
- More to come.

Implementation

New analysis configuration file

Config driven analysis 

Although **everything** can be set in **the single configuration file for the whole analysis chain** the priority in all programs is given to the command line arguments.

Flexibility is also a key.

Implementation

Efficiencies calculation program ✅

- This program calculates the efficiencies used for the spectra corrections.
- **Output**: the ROOT-file with the 2D phase-space histograms and rapidity slices.
- Rapidity slices and efficiencies are defined in the analysis configuration JSON file.
- Works automatically for the defined particles, centralities, rapidity intervals.

Implementation

Efficiencies calculation program

Efficiencies defined in the program

```
# Efficiencies definition: numerator, denominator, name
efficiencies_types_example = [
    # TPC efficiency:
    # Numerator:   MC_PID, RC tracks, Primary by MC.
    # Denominator: MC_PID, MC tracks.
    {'numerator': 'h__eff_counter_3',  'denominator': 'h__eff_counter_0', 'output': 'efficiency_tpc'},
    # Secondaries contamination:
    # Numerator:   MC_PID, Secondary by MC, RC tracks.
    # Denominator: MC_PID, RC tracks.
    {'numerator': 'h__eff_counter_4',  'denominator': 'h__eff_counter_1', 'output': 'contamination_secondaries'},
    # ToF efficiency:
    # Numerator:   MC_PID, has ToF, RC tracks.
    # Denominator: MC_PID, RC tracks.
    {'numerator': 'h__eff_counter_5',  'denominator': 'h__eff_counter_2', 'output': 'efficiency_tof'},
    # MpdPid 'Combined' PID efficiency:
    # Numerator:   RC_PID, has ToF, RC tracks.
    # Denominator: MC_PID, has ToF, RC tracks.
    {'numerator': 'h__eff_counter_9',  'denominator': 'h__eff_counter_5', 'output': 'efficiency_pid'},
    # MpdPid 'Combined' PID purity:
    # Numerator:   RC_PID == PDG, has ToF, RC tracks.
    # Denominator: RC_PID, has ToF, RC tracks.
    {'numerator': 'h__eff_counter_10', 'denominator': 'h__eff_counter_9', 'output': 'purity_pid'},
    # MpdPid 'Combined' PID contamination:
    # Numerator:   RC_PID != PDG, has ToF, RC tracks.
    # Denominator: RC_PID, has ToF, RC tracks.
    {'numerator': 'h__eff_counter_11', 'denominator': 'h__eff_counter_9', 'output': 'contamination_pid'},
    # MpdPid dE/dx only PID efficiency:
    # Numerator:   RC_PID, RC tracks.
    # Denominator: MC_PID, RC tracks.
    {'numerator': 'h__eff_counter_6',  'denominator': 'h__eff_counter_2', 'output': 'efficiency_pid_dedx'},
    # MpdPid dE/dx only PID purity:
    # Numerator:   RC_PID == PDG, RC tracks.
    # Denominator: RC_PID, RC tracks.
    {'numerator': 'h__eff_counter_7',  'denominator': 'h__eff_counter_6', 'output': 'purity_pid_dedx'},
    # MpdPid dE/dx only PID contamination:
    # Numerator:   RC_PID != PDG, RC tracks.
    # Denominator: RC_PID, RC tracks.
    {'numerator': 'h__eff_counter_8',  'denominator': 'h__eff_counter_6', 'output': 'contamination_pid_dedx'}
]
```

Efficiencies defined in the configuration file

```
  ],
"Efficiencies": [
  {"numerator": "h__eff_counter_3",  "denominator": "h__eff_counter_0", "output": "efficiency_tpc"},
  {"numerator": "h__eff_counter_4",  "denominator": "h__eff_counter_1", "output": "contamination_secondaries"},
  {"numerator": "h__eff_counter_5",  "denominator": "h__eff_counter_2", "output": "efficiency_tof"},
  {"numerator": "h__eff_counter_9",  "denominator": "h__eff_counter_5", "output": "efficiency_pid"},
  {"numerator": "h__eff_counter_10", "denominator": "h__eff_counter_9", "output": "purity_pid"},
  {"numerator": "h__eff_counter_11", "denominator": "h__eff_counter_9", "output": "contamination_pid"},
  {"numerator": "h__eff_counter_6",  "denominator": "h__eff_counter_2", "output": "efficiency_pid_dedx"},
  {"numerator": "h__eff_counter_7",  "denominator": "h__eff_counter_6", "output": "purity_pid_dedx"},
  {"numerator": "h__eff_counter_8",  "denominator": "h__eff_counter_6", "output": "contamination_pid_dedx"}
],
```

If efficiencies are not defined in the configuration file those defined in the efficiencies calculation program will be used!

Implementation

Transverse momentum spectra calculation program

- This program calculates the transverse momentum spectra and applies the efficiency and contamination corrections calculated on the previous step.
- **Output**: ROOT-file with the transverse momentum spectra sliced by the defined rapidity bins.
- Three types of the spectra are saved: MC, reconstructed and reconstructed after applying the efficiency and contamination corrections.
- Rapidity slices and efficiencies are defined in the analysis configuration JSON file.
- Works automatically for the defined particles and centralities.

Implementation

Transverse momentum spectra fitting program

- This program fits the transverse momentum spectra calculated with previous program.
- Currently there are only two fit functions:
  - Thermal:

$$\frac{d^2N}{dp_T dy} = \frac{\mathrm{d}N/\mathrm{d}y}{T(m_0 + T)} p_T \exp\left(-\frac{m_t - m_0}{T}\right)$$

  - Blast-Wave:

$$\frac{dN}{p_T dp_T} = C \int_0^{R_{max}} r\, dr\, m_T\, I_0\left(\frac{p_T \sinh\rho(r)}{T}\right) K_1\left(\frac{m_t \cosh\rho(r)}{T}\right)$$

- Initial parameters for the each fit are defined in the JSON files.

Implementation

Transverse momentum spectra fitting program

Thermal fit initial parameters file has the structure of the sublist of the fit parameters for the each:

- particle
  - centrality bin
    - rapidity region

thermal_init.json 12.36 KiB

```
{
  "pim": {
    "0": [
      [96.75, 0.1003, 0.139, 0.2, 1.5],
      [99.27, 0.0977, 0.139, 0.2, 1.5],
      [89.42, 0.1014, 0.139, 0.2, 1.5],
      [85.47, 0.1030, 0.139, 0.2, 1.5],
      [79.28, 0.0972, 0.139, 0.2, 1.5],
      [74.14, 0.1021, 0.139, 0.2, 1.5],
      [70.35, 0.1012, 0.139, 0.2, 1.5],
      [65.70, 0.1008, 0.139, 0.2, 1.5],
      [60.17, 0.0972, 0.139, 0.2, 1.5],
      [53.84, 0.1030, 0.139, 0.2, 1.5]
    ],
```

Parameters: dN/dy, T, m0 (fixed!), pt_low, pt_high, where [pt_low, pt_high] is the fit range.

Implementation

Transverse momentum spectra fitting program

Blast-Wavefit initial parameters file has the structure same for the reconstructed and the Monte-Carlo data (also for the each particle, centrality bin, rapidity interval):

- `C`: Normalization constant. `m0`: Particle rest mass.
- `T`: Kinetic freeze-out temperature. `beta`: Average transverse flow velocity.
- `[pt_low, pt_high]`: Fit range.
- `R_max`, `steps`: Source radius and the number of integration steps.
- `fit_found`: Flag which indicates that fit was found (1) or not (0).

```
[
  3333.33,
  3.728,
  0.15,
  0.4,
  1.0,
  4.0,
  20.0,
  40,
  1
],
```

Implementation

Transverse momentum spectra fitting program

The Blast-Wave (BW) fitting procedure is quite complex and the fit quality is not very often «good».

- The fit is performed with the initial parameters.
- The values of the fit at the low and high transverse momentum bins are compared with the fitted spectra.
- If the difference in both regions is small — the flag `fit_found` is set to 1, the initial parameters are considered as final.
- If the difference in any region is high — the the procedure randomly set the new parameters until "good" parameters are found or until 100 tries. If "good" parameters are found — they overwrite the initial parameters in the fit parameters file, the `fit_found` is set to 1. Otherwise the `fit_found` flag is still 0.
- On the next runs the program will use the fit parameters with the flag `fit_found = 1` and skips the fitting procedure.

Implementation

Transverse momentum spectra fitting program

In this case the **config-driven approach** allows you to target the specific particle, centrality, rapidity interval without affecting other particles and/or regions and thus loosing time to re-fit everything again and again.

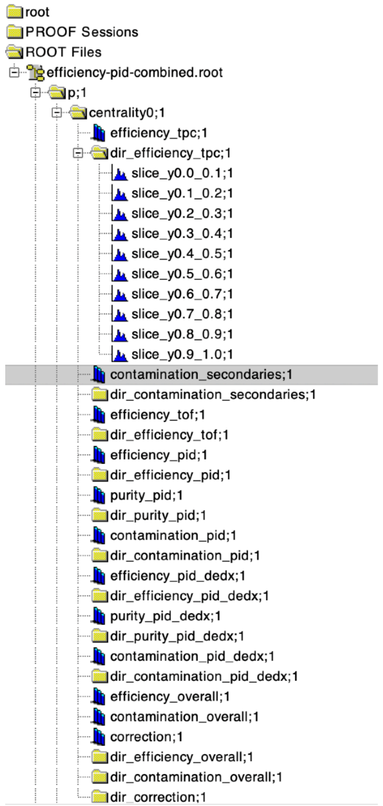
Implementation

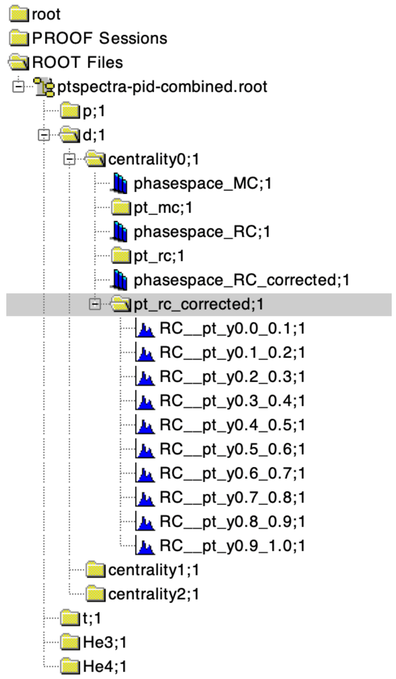
Separate drawing programs

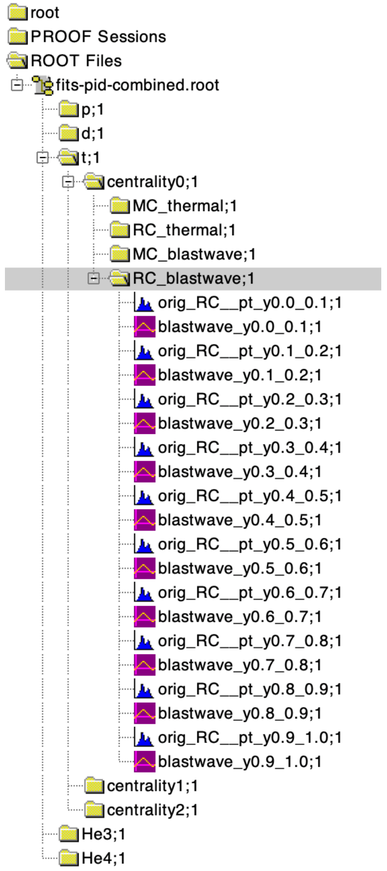
- Efficiency and contamination phase spaces and their slices in the defined rapidity bins.
- Particles phase spaces: Monte-Carlo, reconstructed and reconstructed corrected by the efficiencies and contaminations.
- Transverse momentum spectra: Monte-Carlo and reconstructed (corrected) with the Blast-Wave fits and ratios between the original Monte-Carlo spectra and the reconstructed data + Blast-Wave fits interpolation to the low and high transverse momentum regions.
- Particles dN/dy spectra (Monte-Carlo and reconstructed):
    - The integral of the corrected spectra within defined transverse momentum pT bins.
    - The integral of the interpolation of the Blast-Wave fits to the low and high pT regions.

Output files

- **Clarity, logical structure, self-describing names**







Summary for the first part

- Some old approach ideas works really great, however some modifications were be made.
- The new config-driven analysis approach sounds complex, but in reality is much easier to implement and then to perform than the standard «put everything in the program» method.
- The performance gain becomes even more pronounced when re-use analysis with the tuned fit parameters.

Documentation

and code organisation

Documentation

- Wagon C++ code is documented using the Doxygen engine and tons of comments.
- Analysis and drawing routines are documented with Doxygen-style comments instead of the Python docstring for the sake of unification. Tons of ordinary comments are still included.
- The main wagon C++ code and analysis routines are also supplemented by the documentation in Markdown (README.md files in the repository).
- New documentation dedicated repository organised.

Documentation

New documentation repository: overall view

README.md

Some documentation for/from the PWG-2 group

Getting started

- First steps

Wagons

- Nuclei (documentation)
  - Source code

Analysis

- Nuclei

Created on
April 22, 2025

Documentation

New documentation repository: first steps — newcomers are welcome

0. Install MpdRoot

Instructions here: https://mpdroot.jinr.ru/running-mpdroot-on-the-cluster-nica-lhep-hybrilit/

0.5. Load the helper environment

Load environment for the helper macros:

```
$ . /nica/mpd1/kireev/public/00_env.sh
```

> **Important**
> **Please, note, this environment and helper macros are not official.**
> You can look into their source code for some comments/description to get an idea how to run programs for the each described step.
>
> **Be very careful while using these macros** and even maybe contact me in advance.

1. Run the Monte-Carlo simulations

You can run PHQMD and/or UrQMD models on the NCX batch farm:

`phqmd2batch` : run PHQMD on the cluster

- `-h, --help` show this help message and exit
- `--massta` Target mass (default: 208)

Documentation

New documentation repository: dive deeper into the analysis wagons structure

Please, do not hesitate to contact me if you want to contribute and describe your analysis.

More documentation — better:

- It simplifies onboarding of the newcomers (and students).
- It give insights on what we are doing (pure code does not allow this) — exchange of methods and ideas, cross-checks etc.

MpdNuclei wagon

MPDRoot wagon for the light nuclei analysis. Also applicable for the hadrons: pions, kaons, protons.

Introduction

The main idea of the 'Nuclei' wagon is to collect 2D phase-space hisograms (p_T, y) for the defined particle species. Then the two-dimensional histogram can be 'sliced' by the y bins to get the transverse momentum p_T spectra in these slices. By integrating the spectra one can get the total number of particles in that rapidity bin (slice) and at the end -- rapidity density distribution (dN/dy) by combining the numbers obtained in slices.

Documentation

New documentation repository: analysis in details

Sometimes it is inconvenient to put everything is small details in the code — we can still describe what and how we are doing.

Please, do not hesitate to contact me if you want to contribute and describe your analysis.

More documentation — better:

- It simplifies onboarding of the newcomers (and students).
- It give insights on what we are doing (pure code does not allow this) — exchange of methods and ideas, cross-checks etc.

Postprocessing for the "Nuclei" wagon

Efficiencies in the 'Nuclei' wagon

Common ”event quality” conditions

These cuts are applied to the each event:

- Event has a primary vertex.
- Primary vertex Z-position is within some window (e.g. $|V_Z| < 100$ cm).
- Event has a defined centrality by the TPC particles multiplicity.

Common "track quality" cuts

For the MPD collider mode (used for Bi+Bi @ 9.2 GeV):

- $DCA < 3$ cm
- $N_{hits} > 20$
- The split track is rejected

For the MPD fixed-target mode (**FXT**, used for Xe+W @ 2.5 A.GeV):

- $DCA_{x,y,z} < 1\sigma$
- $N_{hits} > 24$
- The split track is rejected

TPC efficiency

Numerator: reconstructed (actually, MC tracks, associated to the reconstructed) primary tracks which satisfy the reconstruction 'Track quality' condition. Denominator: all simulated Monte-Carlo tracks.

$$\varepsilon_{TPC} = \frac{[\text{Primary(MC)} = \text{TRUE}]\ [\text{Identified(MC)}]\ [\text{Track quality}]}{[\text{Primary(MC)} = \text{TRUE}]\ [\text{Identified(MC)}]} \quad (1)$$

Code organisation

- The wagon and analysis development moved out of the MpdRoot tree:

  now it is easier to implement and check new ideas, refactor code, test the workflow etc.

- The new stand-alone repository is still publicly available.

- The ‘nuclei’ analysis directory is still in the MpdRoot tree:

  when a new «stable» version of wagon code or analysis routines become available it will be immediately synchronised with the MpdRoot ‘dev’ branch.

- Such procedure helps to maintain the development of the analysis active while reducing the commits and merge requests load on the MpdRoot repository: **win-win** approach.

Summary for the second part

- Please, write **documentation**.
- Do not hesitate to put explanatory comments in the code:
  - Describe what, how and why you are doing.
  - Describe input parameters, returned value.
- This sounds annoying, waste of time and unnecessary, but in reality **is extremely important**.

Utilisation

actually, this is the results section

Utilisation

End of the day

Request 37:

5M of PHQMD Xe+W min. bias events at T = 2.5 GeV/n

MPD

Sign Up Log In

Request 37: General-purpose (hadrons, light nuclei), 5M PHQMD Xe+W (T = 2.5 GeV/n, FXT)

Monte-Carlo productions

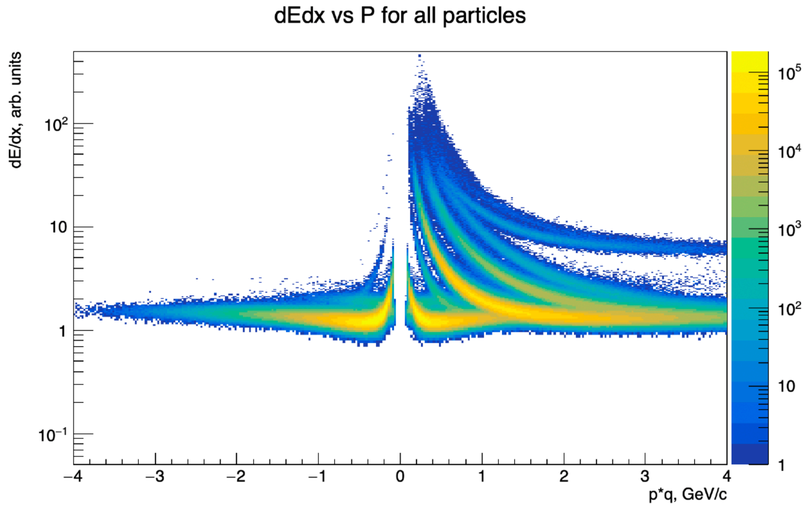
vkireyeu 1 Apr 4

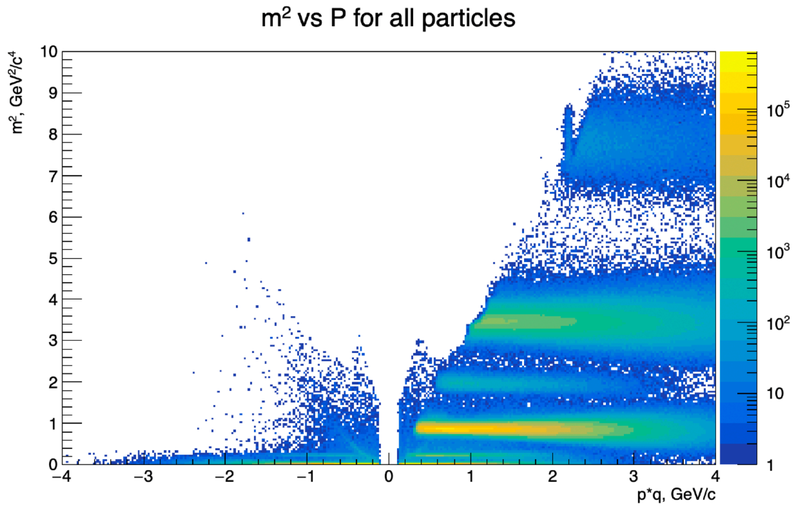
This is a PWG2 general-purpose (hadrons and light nuclei spectra) simulation for min.bias Xe124 + W184 collisions.
Beam kinetic energy: 2.5 A.GeV.
Impact parameter range: 0 - 14.5 fm.
Events: 5M
ECAL excluded to speed-up the calculations.

Apr 4

1 / 2

Apr 4

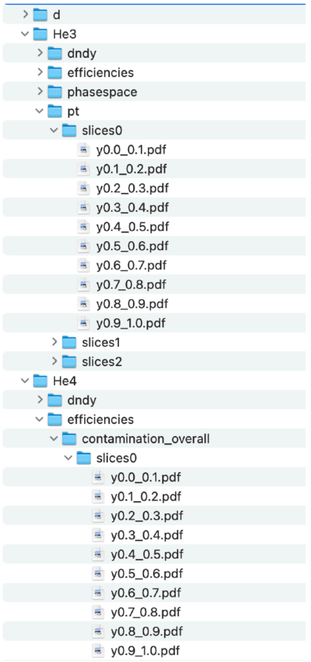


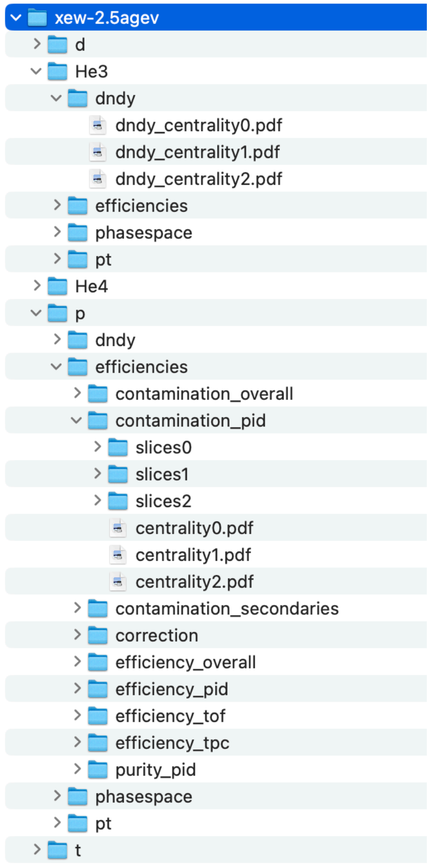


Utilisation

End of the day

- After running the whole analysis chain you obtain **thousands of plots**.
- Here only few are presented to show the proof of work.

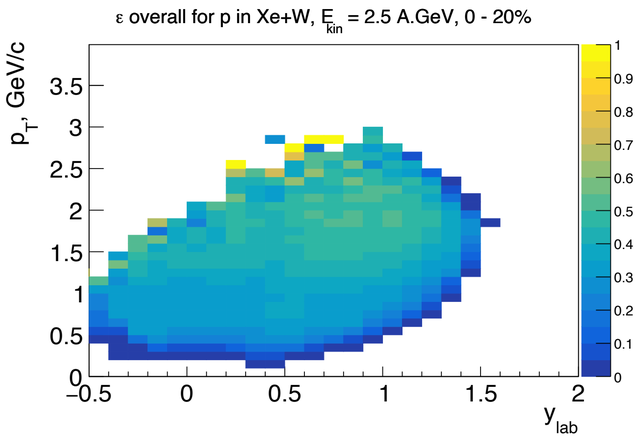


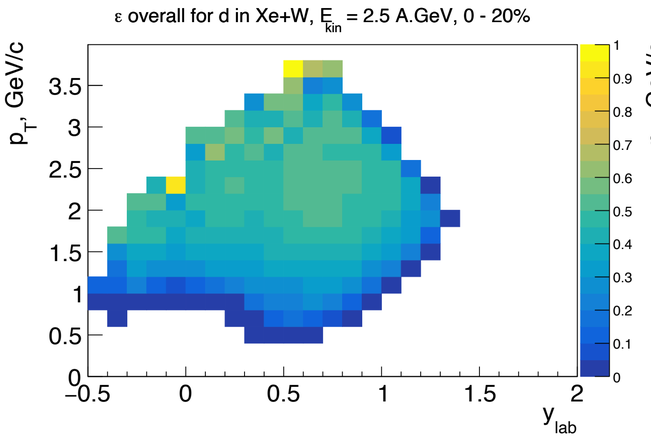


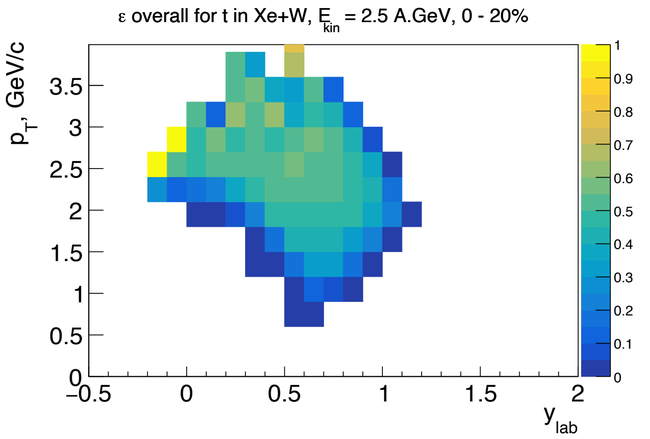

```
$ find . -name *.pdf | wc -l
1695

339 plots per single specie
```

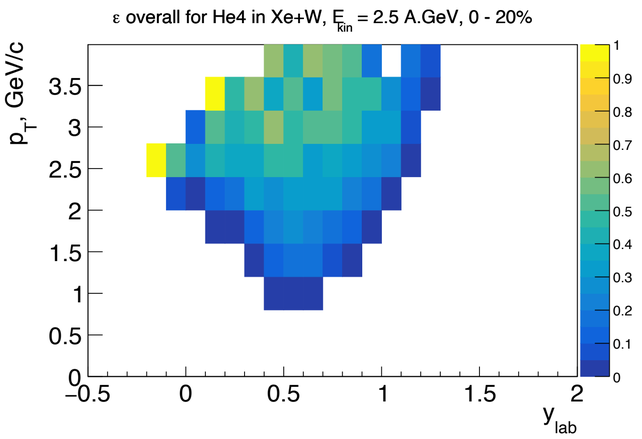
Utilisation

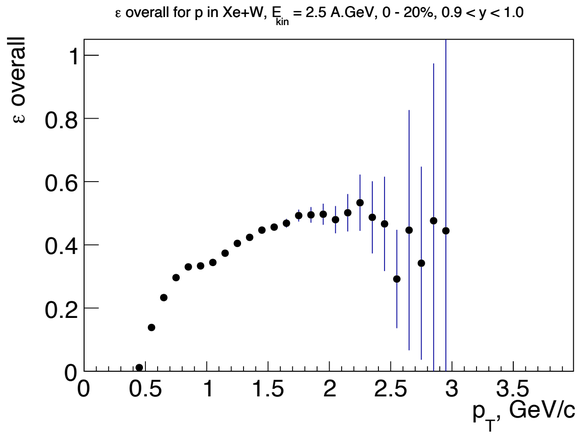
Efficiencies

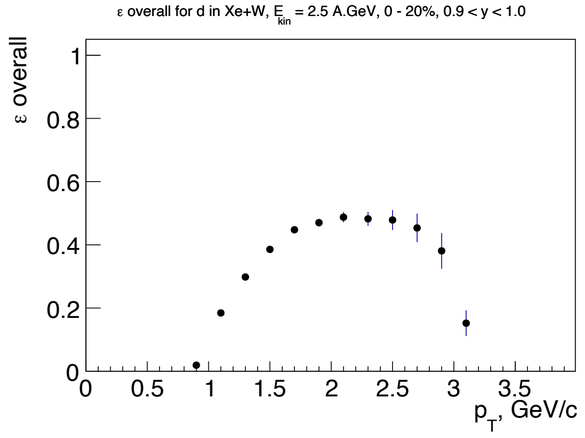


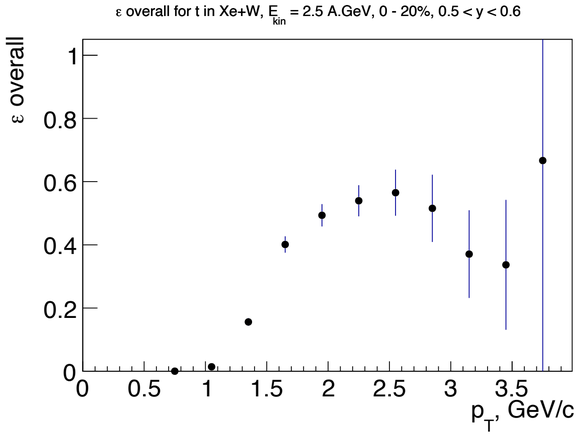


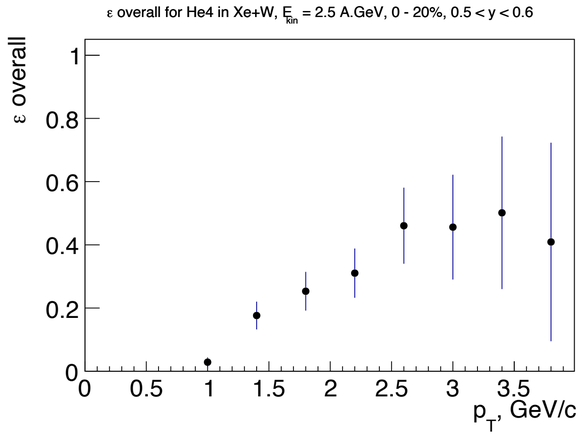






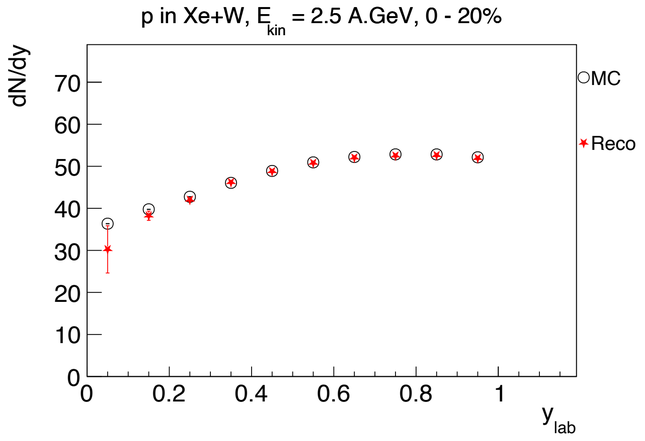


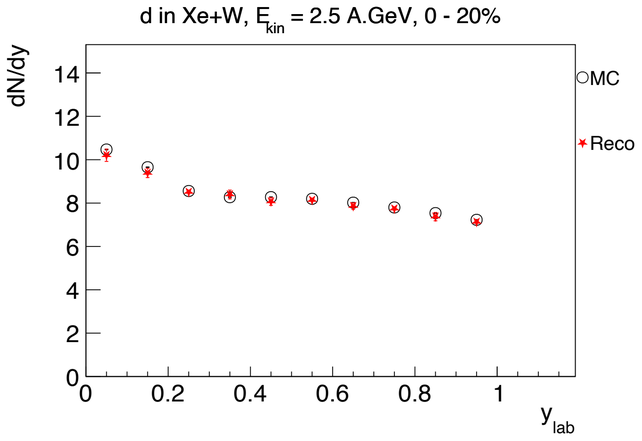


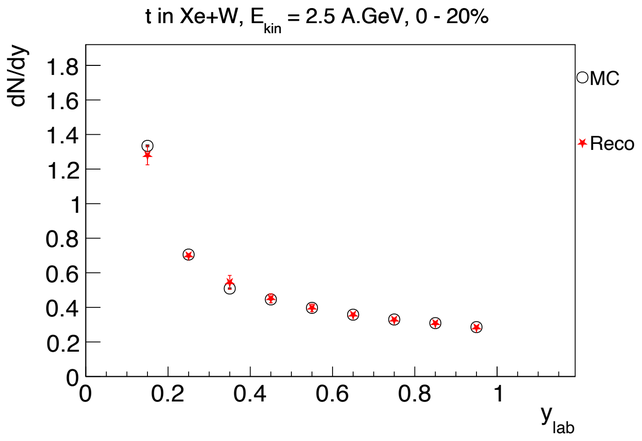


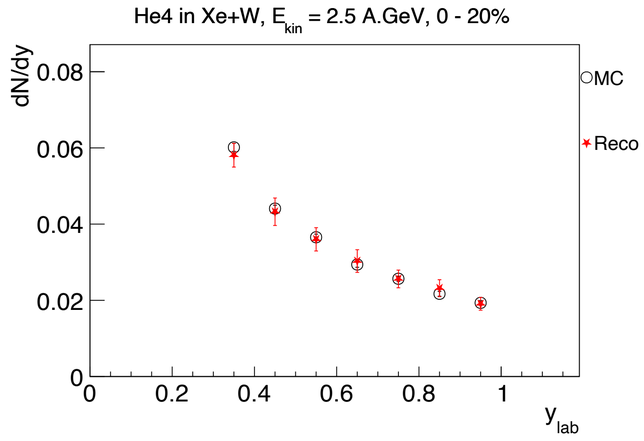
Utilisation

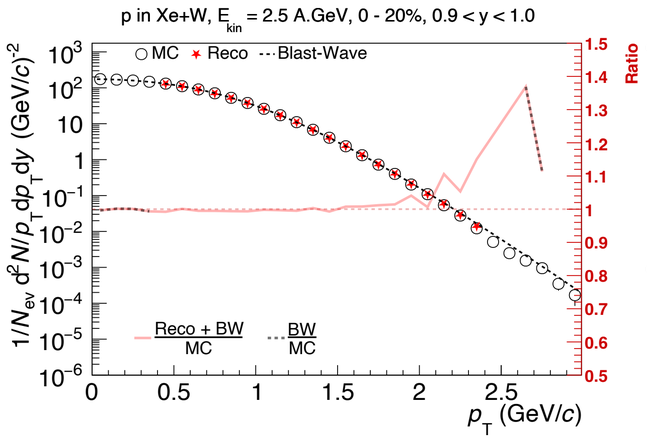
Spectra: p_T, dN/dy

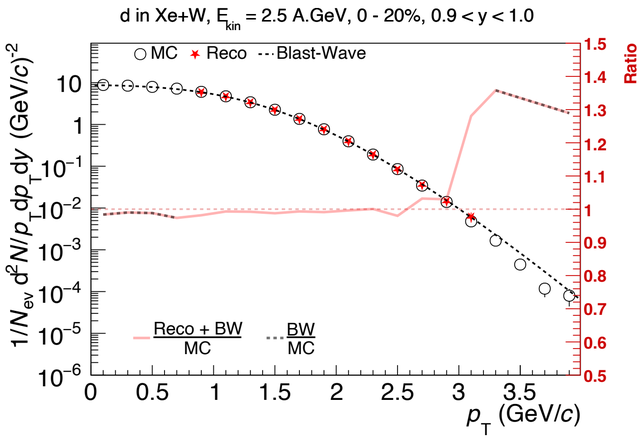


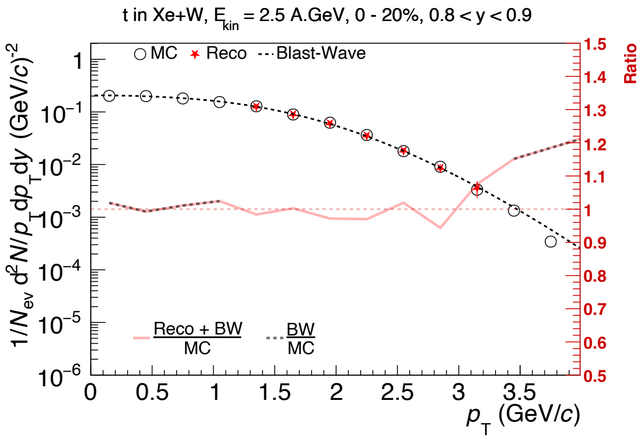


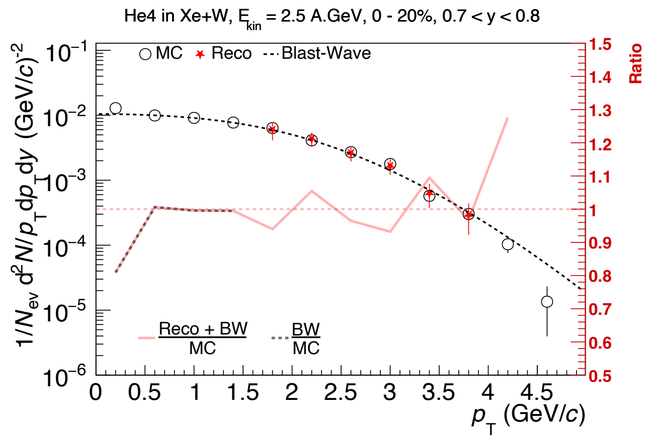












Utilisation

Room for improvement

- PID criteria: purity (contamination), efficiency, low transverse momentum part
- Blast-Wave fit parameters and form: low and high transverse momentum part, dN/dy points.

Sounds easy, but actually can take an indefinite amount of time.

Good news: all machinery is set up.

Summary

this ends soon already

Summary

- We must **think about the future** — students, newcomers — and finally start to write the code in a «good» way:
  - Avoid hardcoded and «magic» numbers.
  - Put explanatory comments in the code.
  - Add the documentation which covers the whole analysis.

It does not take so much time as you can think — but saves your time in the future.

Summary

- The new **config-driven analysis** paradigm was used for the «nuclei» wagon update:
  - Easy to implement.
  - Easy to use.
  - It saves a lot of time during the post-processing (after the MpdRoot train) analysis.

«Put everything in the black box» paradigm is obsolete and dangerous.

Summary

- Despite the right approach used within the «nuclei» wagon, there is room for improvement:
  - PID part in the wagon C++ part must be refactored (reduce the code).
  - More options must be added to the analysis configuration file (post-processing analysis).
  - Small refactor of the post-processing programs for the heavies usage of the configuration file — it is a must.

~1 month to implement and test

Fully config-driven approach