Simulation Data Quality

Automatic checks on monitoring histograms

Marie Sturm[a]

[a] *Humboldt Universität zu Berlin, Germany.*

September, 2023

Abstract

Monitoring of physics variables is crucial to ensure data quality in the LHCb experiment. Automatic tests on monitoring histograms are a well established tool in the online data processing to identify potential issues as they arise. To extend the assurance of data quality for simulation production to the same level as online, we implemented automatic checks on monitoring histograms generated during simulation test jobs. The execution of those checks was incorporated into the GitLab-CI pipeline as well as into the monitoring system *Monet*.

1. Introduction

As the LHC operates at higher and higher luminosities, leading to larger data samples, real-time monitoring of physics variables becomes increasingly vital to maintain data quality within the LHCb experiment. For the Online data quality monitoring (DQM) there are already automatic histogram analysis tools in place, to identify potential issues as they emerge. In Run 3, the LHCb collaboration is expanding these established tools to encompass offline data processing and simulation productions, ensuring data quality to the same level as online.
The primary objective of this project is to extend the existing monitoring capabilities by implementing automatic checks on monitoring histograms to enhance data quality assurance for simulation production. To accomplish this, several key milestones need to be achieved.

1.1. Implementation of significant checks on monitoring histograms

Tests need to be defined that provide meaningful information about data quality.

1.2. Inclusion of analysis in the GitLab CI pipeline

The tests need to be incorporated into the GitLab CI pipeline to ensure that they are executed automatically whenever new productions are added.

1.3. Display test results in Monet

The tests should be included in the monitoring system Monet, making them accessible to the shifters.

1.4. Development of a mechanism to alert shifters in early stage

A mechanism should be implemented to provide the shifters with updates about failed tests and to direct their attention to potential issues as they arise.

Ultimately, the goal is to optimize the utilization of LHCb resources by proactively detecting and addressing potential issues at an early stage. By reinforcing the data-quality assurance practices, the project aims to pave the way for more accurate and reliable physics results during Run 3 and beyond.

2. Monitoring histograms of simulation test jobs

Before new simulation requests are launched, test jobs are run in the Worldwide LHC Computing Grid [2]. The Grid is a collaborative network of computing resources contributed by participating universities, laboratories and institutes to support the LHC experiments. A job refers to a one-time execution of a program, often with the objective of generating or processing a data file.
Once the test jobs are produced the metadata is pushed to the SimDQdata GitLab repository. Each metadata file includes a link to a ROOT file which contains various histograms that monitor different physics variables as can be seen in figure 1.
The ROOT files consist of multiple directories which mainly correspond to the different sub-detectors.

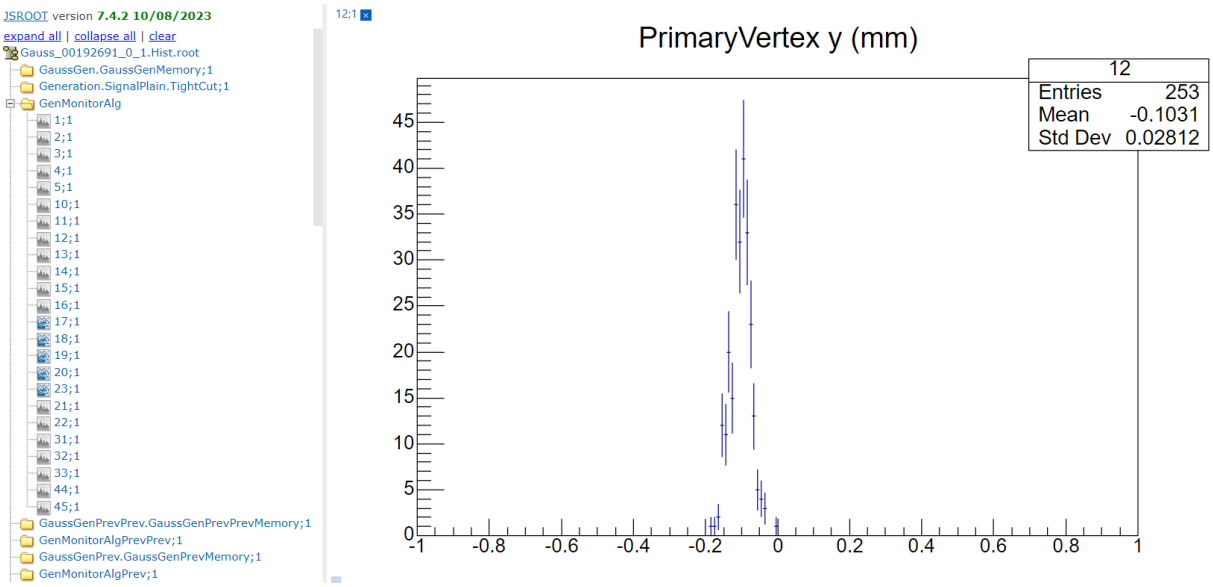

Figure 1: In the simulation test jobs, monitoring histograms of selected physics variables are produced. This image shows one histogram contained in the ROOT file of a specific test job. The histogram represents the distribution of the primary vertex location in y direction. In the left menu one can navigate between directories that mainly correspond to the different sub-detectors.

Because LHCb contains various sub-detectors, corresponding to a vast number of physics variables to monitor, this analysis focuses on setting up the stages 1.1 to 1.4 for the generator phase of the simulation. Once the mechanism is implemented, more checks can be added in the future.

3. Histogram analysis

The implemented algorithm compares each histogram to its respective reference production. The reference samples are appropriate inclusive or minimum-bias samples with the same beam/detector conditions. All 24 histograms from the generator directory are overlaid with their corresponding reference production for quick identification of significant irregularities.
The legend of each histogram provides the results of two tests aimed at quantifying the similarity of the distributions: the Kolmogorow-Smirnow-Test (KS-Test) and the (χ^2)-Test. Those tests help determine if

two datasets are likely to originate from the same underlying population or distribution [3] [1]. A picture of all 24 histograms can be found in the Appendix A.
Furthermore, the algorithm conducts an analysis on three particular histograms representing the primary vertex location in x, y and z direction. Figure 2 shows those plots.

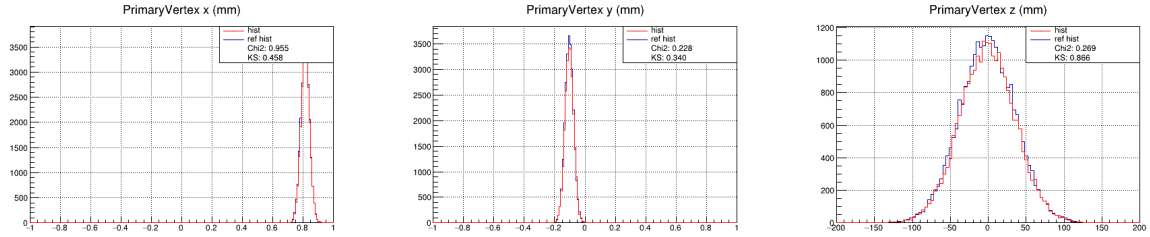

Figure 2: Three of the monitoring histograms that represent the primary vertex location in x, y and z direction. The histograms were plotted on top of their reference production. The legend contains the test result of the KS- and the χ^2-Test

It calculates the mean and standard deviation of the primary vertex histograms. The analysis checks whether those parameters align within predefined tolerance levels with their reference production values. For now, this tolerances were arbitrarily defined and can be changed depending on the required accuracy. In the future, more tests can be implemented, always under the premise to provide meaningful information about data quality.

4. Automatisation

By incorporating the checks into the GitLab-CI pipeline, the testing process becomes automated. Whenever a metadata file is changed or added, the test algorithm automatically runs. Furthermore, when a new merge request is submitted, the tests are conducted for the last ten samples. If any of the checks do not succeed, the test stage reports a failure.
Currently, the checks fail only when the mean or standard deviation deviates more than a predefined tolerance from the reference value. For the χ^2-Test and the KS-Test, no specific tolerance has been defined yet. These tests can be error-prone, particularly when dealing with empty bins and unbinned histograms. Further investigation is required to determine a reasonable tolerance. Before integrating the test results of the KS- and the χ^2-Test into the simulation production workflow, it should be ensured that those tests provide meaningful data quality assurance.

To enhance the identification of failure causes in case the pipeline fails, error messages are included in the unit test jobs. This practice allows the shifters to pinpoint issues without manually sift through entire log files. Additionally, the generated plots are saved as artifacts of the pipeline execution for future reference as can be seen in figure 3.

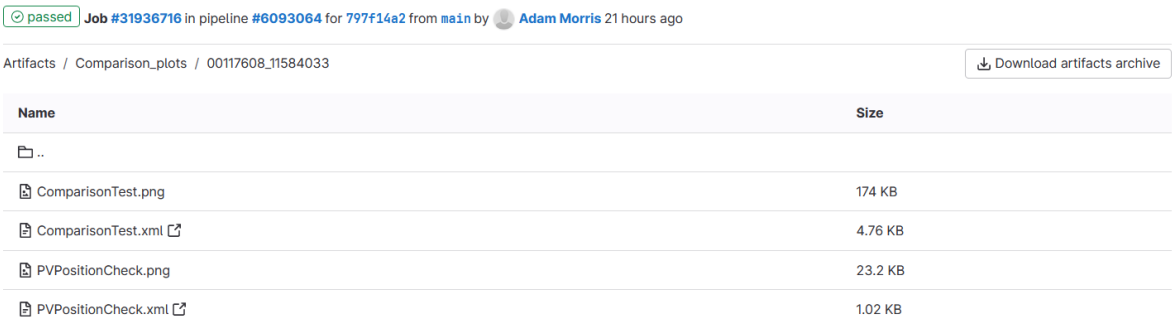

Figure 3: The histograms produced in the automatic tests are saved to the pipeline execution artifacts. The primary vertex histograms are saved in a extra file.

Both the unit test reports and production artifacts can be accessed and reviewed in the *SimDQ* GitLab repository.

5. Including the automatic checks in *Monet*

Monet serves as the central monitoring system employed across various stages of data quality assurance at LHCb, encompassing online, prompt, and simulation DQM. Responsible shifters utilize *Monet* to oversee the functionality of the detectors or the simulation tools, ensuring the reliability of both real and simulated data. Anomalies detected within the histograms can point to potential issues within the experiments or the simulation software.

5.1. Functionality

Currently, *Monet* displays histograms generated from simulation test jobs without supplying further information about data quality, as depicted in Figure 4.

In the left menu, shifters can navigate through various phases of the simulation. The top bar allows the selection of the desired request ID and event type number. Additionally, it provides the option to display the reference histograms for the particular production sample. By clicking on "Rendering Info," users can verify which production file was used as a reference.
Furthermore, *Monet* includes built-in alert tools designed to trigger alarms in response to unusual behavior. However, those tools are not yet set up for the simulation analysis.

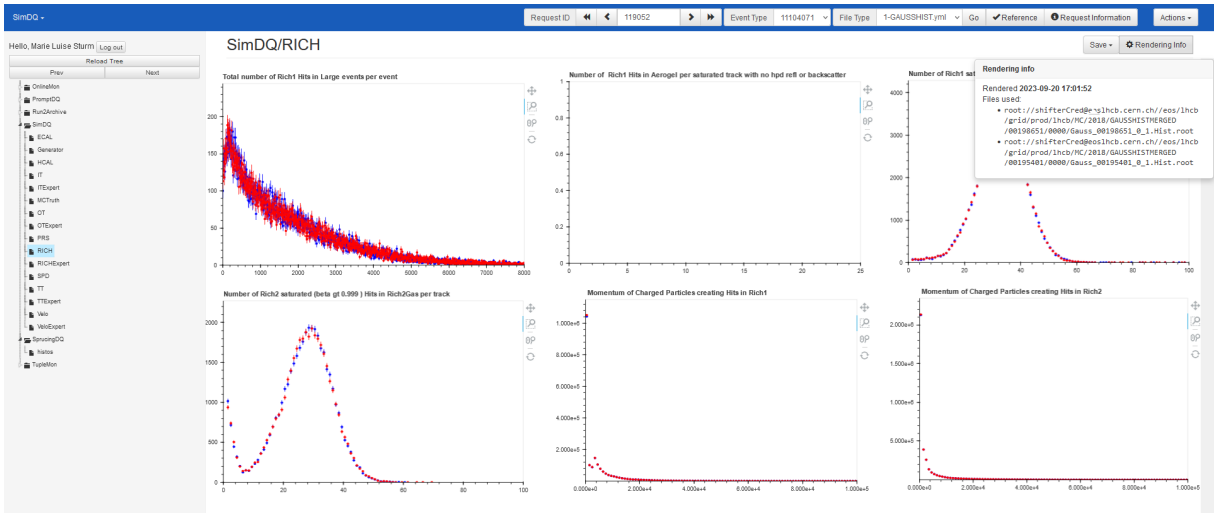

Figure 4: Current state of the *Monet* interface for the simulation monitoring histograms. The menu on the left can be used to navigate between the phases of the simulation. In the top bar the desired event can be specified and the reference production can be either shown or hidden. Which file was used as reference is displayed by clicking on "Rendering Info".

5.2. Repository structure

For the online DQM there is already a structure in place to run analysis functions on the monitoring histograms. Regrettably, this structure of repositories and applications is fairly complex, making the addition of new functionalities a challenging task.
Figure 5 offers a schematic representation of the communication between *Monet* and the Flask applications responsible for analyzing the monitoring histograms.

When a request is made in *Monet*, the system initiates contact with the *HistoYML* GitLab repository to retrieve information about the histograms that should be displayed and what analysis to perform. Subsequently, an API call is sent to *Monitoring Hub*, which, based on the presence of the keyword "taskname", determines whether any analysis should be performed.
If the keyword is found, another API call is dispatched to the *Automatic Analysis* application, which then executes the specified analysis algorithms. The API response received by *Monet* may either contains histogram data alone or additionally includes analysis results.

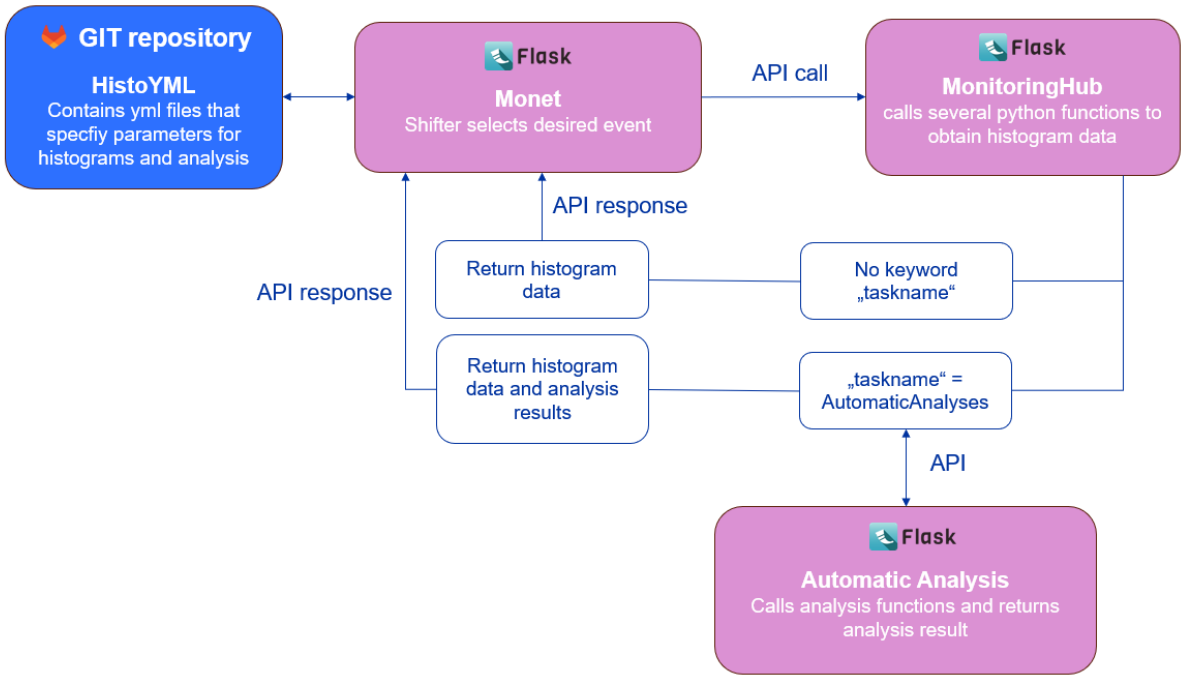

Figure 5: Schematic of the current repository structure. Shifters can select a specific production in the monitoring system *Monet*. *Monet* obtains parameters about the histograms from multiple yaml files contained in the repository HistoYML. *Monet* sends an API call to *Monitoring Hub* , which executes different python functions to obtain the histogram data. Depending on whether a keyword "taskname: AutomaticAnalysis" was supplied for a specific histogram *Monitoring Hub* will either return only the histogram data or send another API to a third flask application *Automatic Analysis*. *Automatic Analysis* runs analysis functions and returns the test results. In this case, both the histogram data and the analysis results are supplied back to *Monet*.

5.3. Implementing test results

To incorporate the automatic checks on the simulation histograms, we built upon the existing structures for the online DQM.
The new test functions have been integrated into *Automatic Analysis*. The yml file in *HistoYML* was modified to include the legends information and details about what analysis should be performed. The snippet of the yml file that contains this information is shown in the Appendix B.
As an analogy to the online histogram analysis, we added the key 'taskname' to the dictionary of each histogram. Currently this key can take two values:

- taskname: AutomaticAnalysis
- taskname: compare_to_ref

Using the first option, any of the already existing functions for the online monitoring histograms can be accessed. The option 'taskname: compare_to_ref' points to the new class of analysis functions.

Furthermore, we introduced a new dictionary, that can be either labeled 'compare_to_ref' or 'analysis'. This dictionary is interpreted by *Monet* to pass the relevant data required for the desired test to *Monitoring Hub*. Depending on the provided keys *Monitoring Hub* invokes the appropriate class of functions in *Automatic Analysis* and returns the analysis results.
In the future there might be consideration to combine *Monitoring Hub* and *Automatic Analysis* to reduce the complexity of the software framework.

In its modified state, the *Monet* page displays the results of the performed tests in the legend of each histogram, as shown in figure 6. This feature provides additional information about the data and simplifies spotting which histograms should be payed attention to.
The four histograms on the left show the results of the KS- and χ^2-Tests in their legends. The legends of the primary vertex histograms contain the results of the mean and standard deviation checks.

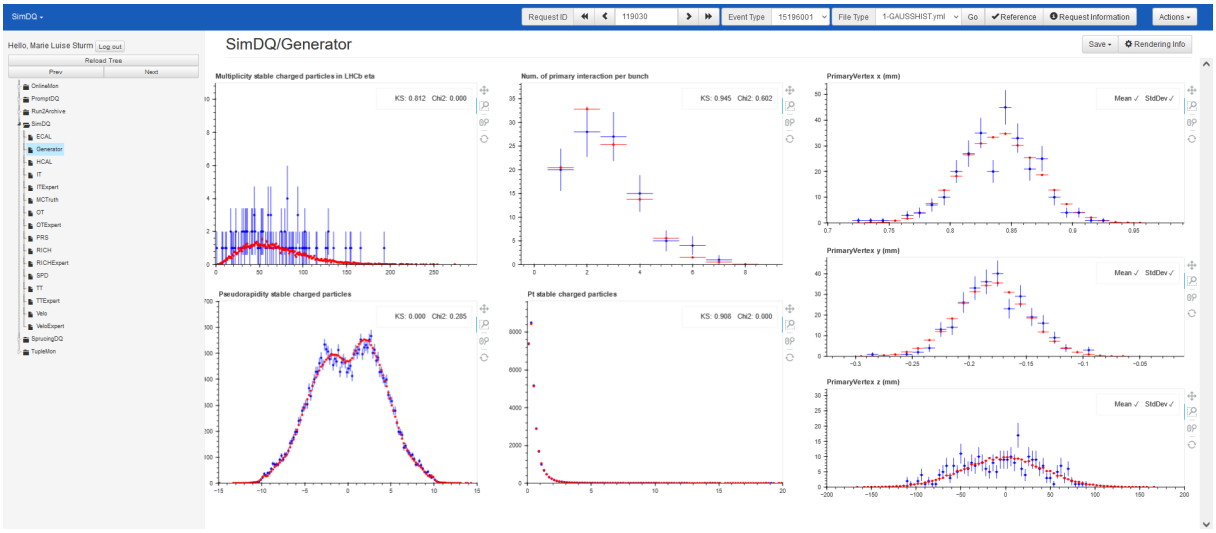

Figure 6: Modified state of the SimDQ *Monet* web page. If necessary, the results of any analysis defined in the *Automatic Analysis* repository can be displayed in the legend of the histograms. In this case, the four histograms on the left contain the results of the KS- and χ^2-test and the legends of the three primary vertex histograms on the right display whether the mean and standard deviation lie within tolerance around the reference value. '✓' means that the calculated value lies within tolerance. '×' indicates that the value is out of tolerance.

6. Conclusion and Outlook

In conclusion, we have successfully integrated automatic checks for various monitoring histograms into the Simulation DQ workflow. The tests have been incorporated into both the GitLab CI pipeline and the monitoring system, *Monet*.
However, the implementation of the automatic checks into *Monet* posed unexpected challenges due to the complex repository structure already in place for online analysis. Simplifying this repository structure could greatly facilitate the implementation of new histogram analysis features in the future.

One milestone of the project, the development of a mechanism to raise alarms if the checks fail, could not be achieved within the time frame. The usability of *Monet* would greatly benefit from the integration of an alert system to identify issues as they arise. This remains a task for the future.
Additionally, as mentioned in the project introduction, for Run 3 there are plans to extend the same tools to offline data processing. This implementation may be smoother now that the framework is established for simulation DQM.

While working on this project, I extended my understanding of integration software development and the tools and technologies that are used in the LHCb experiment.

Acknowledgements

First, I want to thank my supervisors Adam Morris and Nicole Skidmore who were very helpful and encouraging. Furthermore, I am deeply appreciative of the extensive opportunities provided by CERN during my stay. The diversity of the summer student program not only allowed me to delve deeply into the workflows of the LHCb experiment but also encouraged to explore various facets of electrical and mechanical engineering, and computational science, thereby broadening my understanding of physics.

References

[1] *Kolmogorow-smirnow-test*, https://de.wikipedia.org/wiki/Kolmogorow-Smirnow-Test.

[2] B. C. on behalf of the LHCb Simulation & Computing Projects, *Ensuring Simulation Quality in the LHCb experiment*, (2023) https://indico.jlab.org/event/459/contributions/11525/attachments/9360/13567/CHEP2023_Ensuring_Simulation_Quality_in_the_LHCb_with_updates.pdf.

[3] *Reference guide*, Data Analysis Framework ROOT, https://root.cern.ch/doc/master/classTH1.html#ab7d63c7c177ccbf879b5dc31f2311b27.

GitLab repositories

SimDQdata

https://gitlab.cern.ch/lhcb-simulation/simdqdata

Automatic Analysis

https://gitlab.cern.ch/lhcb-monitoring/AutomaticAnalyses

Monitoring Hub

https://gitlab.cern.ch/lhcb-monitoring/MonitoringHub

Monet

https://gitlab.cern.ch/lhcb/Monet

HistoYML

https://gitlab.cern.ch/lhcb/histoyml

Web pages

Monet

https://lbwebmonet.cern.ch/

Appendix A. Monitoring Histograms

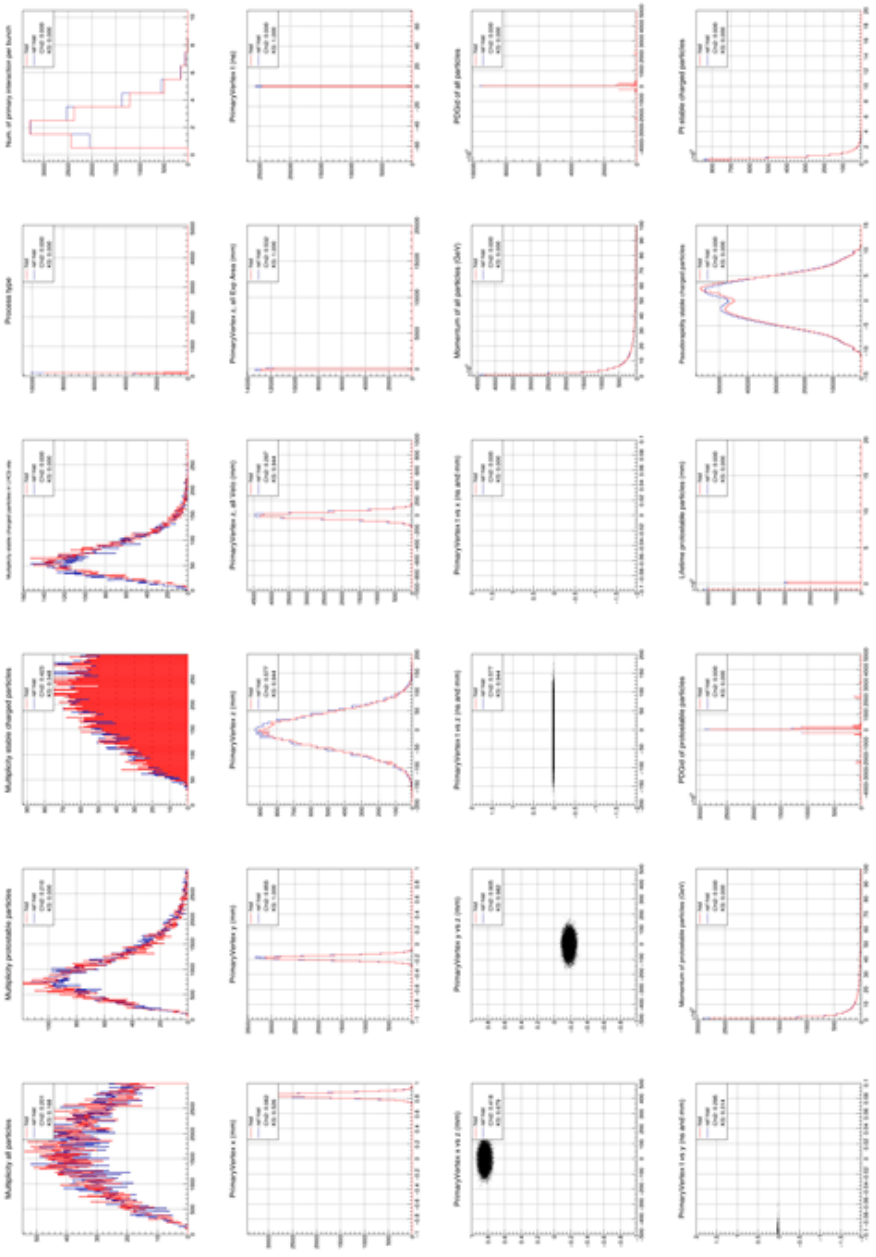

Figure A.7: All 24 monitoring histograms of the generator phase are plotted together with their reference production. The legend of each histogram contains the results of the Kolmogorov-Smirnow- and the χ^2-Test as explained in section 3.

Appendix B. Section added to the *Generator.yml* file in *HistoYML*

```
display_options:
  analysisresults: legend
  stats: '000000000'
compare_to_ref:
  type: CheckMeanStdDevRef
  inputs:
      - name: GenMonitorAlg/12
taskname: compare_to_ref
```

Listing 1: Section that was added to the file in *HistoYML* as explained in section 5.3. The dictionary 'display_options' specifies that only the analysis results should be shown in the legend. The mapping 'compare_to_ref' contains information about the analysis that should be performed. The key 'type' points to a specific function name. The key 'inputs' determines which histogram is used for the analysis. The option 'taskname: compare_to_ref' indicates that the class of functions that compares the histograms to its reference production should be used.