JOINT INSTITUTE FOR NUCLEAR RESEARCH

Veksler and Baldin laboratory of High Energy Physics

FINAL REPORT ON THE START PROGRAMME

BMN AI Health System — Milestone Toward Real-Time Smart Monitoring and Anomaly Detection at BM@N

Supervisor:

Dr. Ilnur Gabdrakhmanov

Student:

Fida Ben Zaghdane

Higher Institute Of Computer Science Of Mahdia, Tunisia

Participation period:

July 27 – September 20,

Summer Session 2025

Dubna, 2025

TABLE OF CONTENTS

ABSTRACT

The BMN AI Health System is a modular offline platform for detector health monitoring in the BM@N experiment at JINR. It implements Principal Component Analysis (PCA) for multivariate anomaly detection, reinforced by physics-informed validation and BMNROOT-style outputs [1]. In the current implementation, the system integrates directly with ROOT data using PyROOT [2, 3], employs intelligent branch mapping to accommodate run-to-run schema variability, and generates histograms via robust custom detector analysis that preserves BMNROOT conventions while avoiding macro brittleness. The platform produces deterministic, reproducible artifacts (HTML report, JSON results, performance metrics) and is engineered with a layered architecture that is intentionally extensible toward non-linear models, real-time online monitoring, and distributed microservices. This work delivers a practical, operator-ready baseline while establishing credible paths for future evolution aligned with HEP software best practices. In practice, grouped PCA combined with physics-informed validation enables interpretable, operator-trustworthy monitoring today, and the same models and thresholds are portable to the future online/edge pathway (see Section 2.1.1), ensuring continuity from offline baseline to real-time operations.

CHAPTER 1 INTRODUCTION: CONTEXT AND MOTIVATION

1.1 The BM@N Experiment within the NICA Project

The BM@N (Baryonic Matter at Nuclotron) experiment is a cornerstone of the NICA megascience project at the Joint Institute for Nuclear Research (JINR). Its primary scientific objective is to explore the properties of dense baryonic matter, a state believed to occur in the cores of neutron stars and during core-collapse supernovae. By studying heavy-ion interactions with fixed targets at 1–6 A GeV, BM@N aims to map the QCD phase diagram—an area of central importance to nuclear and particle physics. The forward spectrometer comprises silicon micro-strip tracking detectors (SiBT, SiProf, FSD) positioned near the target for precise vertex reconstruction; Gaseous Electron Multipliers (GEM) for high-rate tracking within the magnetic field; Cathode Strip Chambers (CSC) for precision tracking downstream of the magnet; segmented time-of-flight systems (TOF400/TOF700) for timing and particle identification; the Forward Hadron Calorimeter (FHCal) for centrality and reaction-plane measurements; and fast trigger detectors (BD, SiMD). This multi-detector system produces high-dimensional, heterogeneous data (waveforms, amplitudes, times, positions, energies) at high rate, motivating automated, ML-assisted detector health monitoring [4] [5].

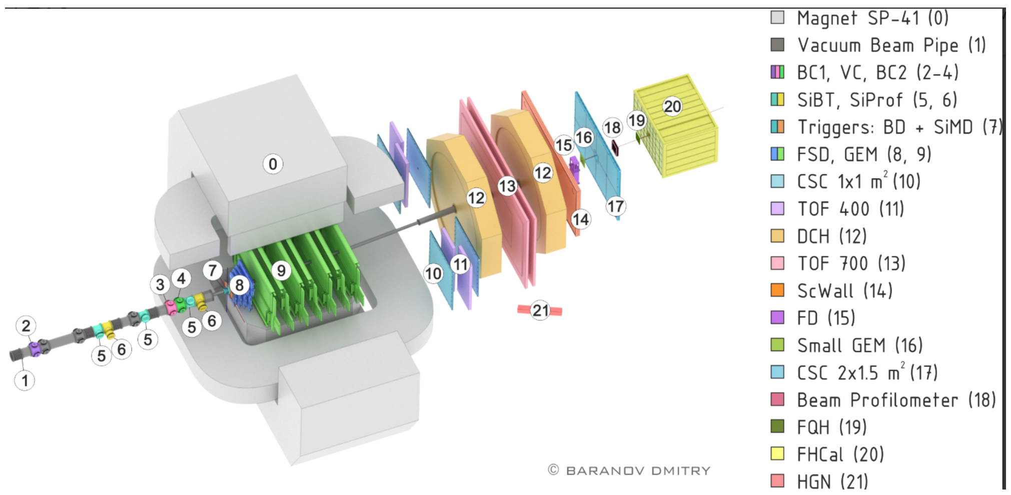

Figure 1.1: BM@N detector installation at JINR. Source: BM@N Collaboration [4].

- **Near-target tracking**: Silicon detectors (SiBT, SiProf, FSD) for vertex reconstruction
- **Magnetic field tracking**: GEM chambers for high-rate particle tracking
- **Downstream tracking**: CSC chambers and Drift Chambers (DCH)
- **Timing systems**: TOF400/TOF700 for particle identification and time-of-flight
- **Trigger systems**: BD+SiMD, BC1/BC2, ScWall for event selection
- **Forward detection**: Forward Detector (FD) and FHCal for energy/centrality

1.2 The Challenge of Detector Health Monitoring in HEP

Ensuring reliable detector operation in high-energy physics is formidable. Conventional QA workflows already employ a rich set of diagnostics beyond one-dimensional histograms, including 2D occupancy maps, beam profile plots, correlation panels (e.g., time vs. amplitude/strip), and rates versus run conditions [2]. However, manual inspection of these views does not scale with today's data volumes, detector heterogeneity, and the need for rapid feedback. Subtle, multi-detector and multivariate effects can be missed when plots are reviewed in isolation, and threshold tuning is often ad-hoc and not reproducible.

The community is therefore moving toward data-driven monitoring—combining statistical models with domain-faithful visualization—as evidenced by large-scale efforts at the LHC. The BMN AI Health System aims to automate and enhance QA using machine learning while retaining interpretability and physics credibility through validation layers, standardized artifacts, and BMNROOT-style outputs.[2, 3] [6].

1.3 Project Objectives and Contributions

This project delivers a modular offline monitoring layer for BM@N that combines PCA-based multivariate anomaly detection with physics-informed validation and correlation analysis. It produces operator-ready, reproducible HTML reports presenting metrics, anomaly scores, and visual evidence. The system implements a complete pipeline from ROOT ingestion (PyROOT) to final report generation, with intelligent branch mapping to handle schema variability. Domain-faithful histograms are produced via robust custom detector analysis integrated with the BMNROOT ecosystem, strengthening operator trust. The architecture is performance-aware and extensible, establishing a foundation for non-linear models and future online monitoring [2, 3] [1] [6].

Acronyms

- PCA: Principal Component Analysis
- GIL: Global Interpreter Lock (Python)
- HEP: High Energy Physics
- QA: Quality Assurance (BMNROOT QA)
- BMNROOT: BM@N ROOT-based software framework
- PyROOT: Python bindings to ROOT [2, 3]
- PNG/JSON: Portable Network Graphics / JavaScript Object Notation

CHAPTER 2 SYSTEM ARCHITECTURE AND DESIGN PHILOSOPHY

2.1 Project Goals

- Deliver a reliable, reproducible offline monitoring baseline for BM@N.
- Detect multivariate anomalies via PCA coherence while maintaining interpretability.
- Validate ML indications via physics-informed rules to reduce false positives.
- Provide BMNROOT-faithful visual outputs to support operator trust.
- Establish an architecture extensible to non-linear methods and future online pipelines.

2.2 Layered Architecture Overview

The BMN AI Health System is designed with a layered architecture to ensure modularity, scalability, and extensibility. This structural approach allows for independent development and modification of each component, which is crucial for a project with an ambitious long-term roadmap. Conceptually, the architecture comprises four layers, each with a specific responsibility. The Presentation Layer handles all user-facing outputs, including the generation of final HTML reports and performance metrics, as well as robust logging for run diagnostics. The Business Logic Layer constitutes the analytical core of the system, containing the PCA coherence monitor, the physics validator, the histogram analyzer, and the correlation analysis modules. The Data Processing Layer is responsible for data ingestion from ROOT files (PyROOT [2, 3]), feature extraction, data quality gating, and vectorization, which prepares the raw data for analysis. Finally, the Integration Layer provides foundational services, including the BMNROOT QA bridge and intelligent branch mapping, together with adaptive controls for worker count and event caps.

This layered design is not merely a software engineering formality; it anticipates the system's future evolution. The Data Processing and Business Logic layers can be viewed as the core computational engine which, in a future online implementation, could be deployed as a distributed "edge" computing platform near detector readout systems for minimal latency. More computationally intensive tasks—such as training or re-calibration—can be offloaded to centralized HPC resources. Finally, the Integration Layer provides foundational services, including the BMNROOT QA bridge [1] and intelligent branch mapping, with adaptive controls for worker count and event caps; BMNROOT is built atop the FAIRRoot experiment framework [7]. This design enables a graceful transition from an offline, batch-processing system to a low-latency, real-time alert system while respecting HEP constraints (e.g., Python GIL with PyROOT I/O; see Section 5.2).

2.3 Core Modules and Functional Mapping

The system's modularity is realized through a clear mapping of functionality to Python modules. Execution is orchestrated by `src/offline_analysis/offline_data_analyzer.py`, which coordinates the flow of data through the pipeline. ROOT ingestion and schema-tolerant access are handled via PyROOT [2, 3] inside the analyzer and the `src/data_extraction/digi_data_extractor.py` module. The core analytical work is summarized below.

Table 2.1: Module map: files to responsibilities.

`Module / Path`	**Responsibility**
`src/offline_analysis/offline_data_analyzer.py`	Orchestrates pipeline; coordinates ingestion, analysis, validation, reporting.
`src/data_extraction/digi_data_extractor.py`	ROOT ingestion via PyROOT [2, 3]; schema-tolerant access.
`src/analysis/pca_coherence_monitor.py`	PCA-based multivariate monitoring (scikit-learn [6]).
`src/ml/enhanced_pca_monitor.py`	Non-linear prototypes (KernelPCA / Autoencoders).
`src/core/raw_data_correlations.py`	Correlation analysis (Pearson / Spearman).
`src/data_extraction/digi_branch_detector.py` `config/branch_mapping_config.yaml`	Intelligent branch mapping; detector-specific candidates/patterns; caching.
`src/integration/bmnroot_qa_bridge.py`	BMNROOT QA bridge and orchestration [1].
`src/integration/bmnroot_custom_analysis.py`	BMNROOT-faithful histogram generation without macro dependency.

CHAPTER 3 DATA PROCESSING AND FEATURE ENGINEERING

3.1 The ROOT Ingestion and Feature Extraction Pipeline

ROOT [2] is CERN's object-oriented data analysis framework for HEP; PyROOT [2, 3] provides Python bindings to ROOT so we can access ROOT I/O and classes from Python.

The pipeline begins with ROOT file ingestion via PyROOT in batch mode [2, 3]. Feature extraction converts detector signals (e.g., amplitude, timing, coordinates) into structured, tabular data suitable for analysis. Group-wise normalization accounts for intrinsic detector response differences, enabling detection of subtle behavioral changes that raw values would obscure. The system also computes cross-detector features to surface systemic anomalies (e.g., timing offsets shared across subsystems).

To mitigate per-run schema variability ("data schema drift"), intelligent branch mapping detects and adapts to detector branch names using configurable candidates and patterns. Defensive accessors check branch existence, and missing branches are handled gracefully.

3.2 Correlation and Physics-Informed Validation

After feature extraction, correlation analysis (e.g., Pearson/Spearman) is applied for interpretable inter-detector relationships, supporting operator trust.

Physics-informed validation enforces domain rules (timing windows, amplitude/energy constraints, cross-detector deltas), filtering ML-flagged anomalies that are not physically meaningful and reducing false positives in a high-stakes environment.

CHAPTER 4 METHODS

4.1 Software Architecture (Layered)

In simple terms, the system has four parts: we present results to the user, analyze data, prepare the data, and integrate with BMNROOT. The diagram below shows how these parts connect.

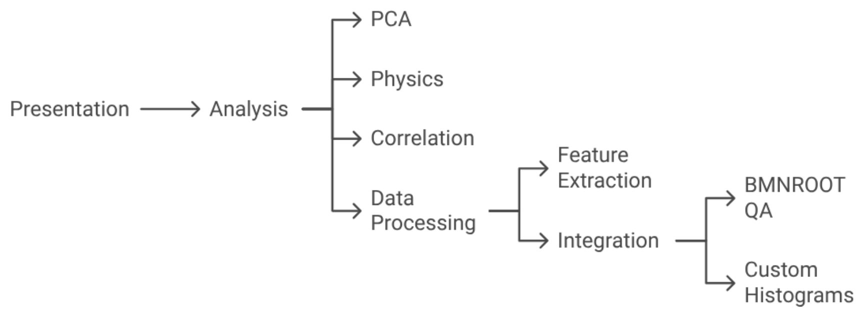

Figure 4.1: Layered software architecture showing presentation, analysis, data processing, and integration components.

Key modules (for orientation): `src/offline_analysis/offline_data_analyzer.py`, `src/analysis/pca_coherence_monitor.py`, `src/ml/enhanced_pca_monitor.py`, `src/integration/bmnroot_qa_bridge.py`, `src/integration/bmnroot_custom_analysis.py`, `src/data_extraction/digi_branch_detector.py`.

4.2 Processing Mechanics & Parallelism

The Python Global Interpreter Lock (GIL) [8] constrains concurrent Python bytecode, so ROOT I/O is executed sequentially via PyROOT [2, 3]. After ingestion, CPU-only phases are parallelized safely: feature extraction, PCA transforms, and correlation computations run in worker processes. Parallelism is configurable to match the machine and workload:

- `processing.enable_multiprocessing`: turn the worker pool on/off
- `max_workers`: cap number of workers (system-aware default if omitted)
- `max_events_per_file`: bound per-file load to stabilize latency/memory

PCA model fitting is performed offline once on a healthy "platinum" run, while per-run projections (transforms) are parallelized. This design preserves GIL safety, avoids PyROOT concurrency hazards, and yields predictable end-to-end runtimes on multi-core systems.

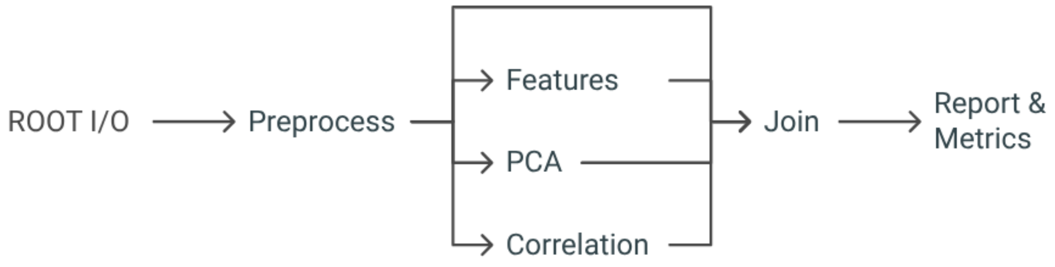

Figure 4.2: Processing model—sequential ROOT I/O followed by parallel CPU-only analysis fan-out.

4.3 PCA-Based Monitoring

PCA [6] is used to monitor groups of detectors with two complementary scores: Hotelling's T^2 (extremeness within the PCA subspace) and the Q-residual (squared reconstruction error, i.e., off-subspace deviation). Using both reduces blind spots.

What T^2 and Q mean (brief)

For a standardized feature vector $\mathbf{x}$ (zero mean, unit variance per feature), PCA yields scores t_i along retained components with variances λ_i.

$$T^2 = \sum_i \frac{t_i^2}{\lambda_i} \qquad \text{(large } T^2 = \text{unusual but still “on-model”)}$$

$$Q = \|\mathbf{x} - \widehat{\mathbf{x}}\|_2^2 \qquad \text{(large } Q = \text{structure not captured by the model)}$$

where $\widehat{\mathbf{x}}$ is the projection back from the PCA subspace.

How it is implemented in this system

- We standardize features per detector group (e.g., TOF, Silicons) and compute T^2 and Q for each event in that group (scikit-learn [6]). - Grouping keeps models small and stable under slow drifts; scoring is fast and parallelized in the CPU-only stage. - PCA fitting is done once on a healthy “golden” run; production runs only apply the transform and scoring.

Thresholds and alerting (concise)

- For each detector group d, we take all eventwise scores from a healthy run and set per-group thresholds as high quantiles (e.g., 99.5

$$\theta_d^{(Q)} = \text{quantile}_{0.995}\{Q_{i,d}\} \quad \text{and} \quad \theta_d^{(T^2)} = \text{quantile}_{0.995}\{T_{i,d}^2\}.$$

- During monitoring, an event i in group d is flagged if $Q_{i,d} > \theta_d^{(Q)}$ or $T_{i,d}^2 > \theta_d^{(T^2)}$. - We also report per-run summaries (e.g., fraction of flagged events per group) and apply physics validation to gate alarms.

Why this works in practice

- Robust features and per-group standardization keep inputs stable; emerging faults tend to increase Q even when distributions are non-Gaussian. - High-dimensional detector spaces are compressed to a few components (typically 2–3) that capture dominant modes, enabling compact monitoring and faster checks.

4.4 BMNROOT Integration & Histograms

BMNROOT QA [1] refers to BMNROOT's quality-assurance conventions and tooling for detector histogramming and monitoring. The underlying experiment framework is FAIRRoot [7]. The QA bridge handles detector enumeration, branch detection, and calls our custom analysis to generate BMNROOT-style histograms [1]. This avoids fragile macro execution while preserving domain-faithful outputs, e.g., `tof700_run8199_timel.png`, `tof400_run8199_time.png`, `bd_run8199_amplitude.png`, `csc_run8199_strips.png`.

4.5 Technical Software Details

At a glance: the analyzer (`src/offline_analysis/offline_data_analyzer.py`) orchestrates ingestion, features, analysis, and reporting; PCA lives in `src/analysis/pca_coherence_monitor.py` (with non-linear prototypes in `src/ml/enhanced_pca_monitor.py`); BMNROOT integration and custom histograms are in `src/integration/bmnroot_qa_bridge.py` and `src/integration/bmnroot_custom_analysis.py`; branch detection is handled by `src/data_extraction/digi_branch_detector.py` with `config/branch_mapping_config.yaml`.

Environment (example): Python 3.12; ROOT 6.x with PyROOT [2, 3] and RDataFrame [9]; NumPy [10], pandas [11], scikit-learn [6], Matplotlib [12]; BMNROOT [1] at the local installation (record commit/tag on publication). Underlying frameworks: FAIRRoot for experiment infrastructure [7] and BMNROOT for detector-specific reconstruction and QA [1].

4.6 Reproducibility Framework

We prioritize deterministic, repeatable runs. Configuration is YAML-based and versionable; key toggles (e.g., `processing.max_events_per_file`, detector enable flags, branch patterns) are captured in config files. Each run produces a consistent set of artifacts:

- `analysis_report.html` – human-readable, self-contained report
- `analysis_results.json` – structured machine-readable summary
- `performance_metrics.json` – timing/throughput/memory metrics

- `histograms/*.png` – per-detector visual evidence

Given the same input file, environment, and configuration, outputs are consistent run-to-run. This approach aligns with open science practices in HEP (cf. CERN ROOT and reproducible workflows) [2, 9].

CHAPTER 5 PROBLEM ENCOUNTERED AND RESULTS

5.1 Problems Encountered and Strategic Mitigations

The system addressed several engineering challenges during development. The table summarizes root causes and implemented mitigations.

Table 5.1: Key problems and concise mitigations.

Problem	Mitigation
PyROOT Concurrency Hazards	Serialize ROOT I/O; parallelize only CPU-only phases via a process pool; avoid mixed I/O in workers [8, 2].
Heterogeneous Detector Branches	Intelligent branch mapping (candidates/patterns), defensive accessors, and graceful fallbacks.
Histogram Blind Spots & Operator Trust	Use PCA coherence as the multivariate signal; require physics-informed validation before alerts; include BMNROOT-style histograms as human-readable evidence.
Threshold Tuning Across Detectors	Per-group scaling with data-driven thresholds (e.g., $\mu + k\sigma$) derived from a healthy reference run.
Memory and Long-tail Latency	Chunked extraction, vectorized operations, and configurable event caps to bound RAM and stabilize latency.

PyROOT Concurrency Hazards

ROOT is optimized for multithreading, whereas Python's GIL allows only one thread to execute Python bytecode at a time. To avoid deadlocks and race conditions, we enforce sequential ROOT I/O and isolate parallelism to CPU-only phases (feature engineering, PCA transforms, correlations). Where higher parallelism is needed, multiprocessing provides separate interpreters/processes to bypass GIL constraints safely [8, 2].

5.2 Results

5.2.1 Functional Outcomes

On run 8199, the system analyzed priority detectors (TOF400, TOF700, BD, CSC) and produced BMNROOT-style PNG histograms for each. Intelligent branch mapping correctly identified detector branches present in the file; FHCal was processed conditionally depending on branch availability. The HTML report embedded per-detector galleries and summarized metrics, and all artifacts were reproducible given the same input and configuration.

5.2.2 ROOT Integration Outcomes

The PyROOT integration operated in batch mode (no GUI), with defensive branch access and schema-tolerant handling [2, 3]. The BMNROOT QA bridge invoked robust custom analysis routines to generate histograms while preserving naming conventions and styles familiar to operators [1]. This approach avoided brittle macro execution and stabilized histogram generation across runs. Standardized outputs included, for example: `tof700_run8199_timel.png`, `tof400_run8199_time.png`, `bd_run8199_amplitude.png`, `csc_run8199_strips.png`.

5.2.3 Batch and Parallel Processing Behavior

The system intentionally serializes ROOT I/O (to remain GIL-safe) and parallelizes CPU-only phases (feature engineering, PCA transforms, correlations) via worker fan-out. In the tested configuration (e.g., `processing.max_events_per_file: 1000`, single file), end-to-end processing was observed on the order of tens of seconds, with stable behavior and no deadlocks. Throughput can be increased by:

- Raising `max_events_per_file` where appropriate;
- Enabling `enable_multiprocessing` and increasing `max_workers` for CPU-only stages;
- Batching multiple input files (sequential I/O; overlapping CPU analysis).

This design provides predictable scaling characteristics while respecting PyROOT's threading constraints [2, 3].

5.2.4 System Flexibility Across Versions

The pipeline remained robust across environment variations by:

- Using PyROOT APIs compatible with ROOT 6.x [2, 9];
- Externalizing BMNROOT location via configuration (`bmnroot_path`);
- Detecting detector branches dynamically based on configured candidates/patterns;
- Keeping outputs and naming standardized independent of minor schema changes.

These measures allow the same analysis to run against different data-taking periods and local installations with minimal reconfiguration.

5.2.5 Foundation for Advanced AI Integration

The current results establish a dependable baseline for future, more advanced techniques. The system delivers:

- Clean, standardized features and per-detector summaries suitable for model training;
- Deterministic artifacts (report/JSON/metrics) for auditability and benchmarking;
- Per-detector histogram assets for hybrid human-in-the-loop validation;
- Modular monitors where non-linear methods (KernelPCA/Autoencoders) can be enabled once calibrated [6].

Together, these outcomes form a practical foundation to evaluate and integrate next-step AI models, and to evolve toward an online monitoring pipeline while preserving interpretability.

5.2.6 Artifacts

- `enhanced_analysis_results/analysis_report.html`
- `enhanced_analysis_results/analysis_results.json`
- `enhanced_analysis_results/performance_metrics.json`
- `enhanced_analysis_results/histograms/*.png`

5.2.7 Reliability & Performance (qualitative)

- Removing external macro execution eliminated common failure modes and made histogram generation reliable.
- Serial ROOT I/O combined with parallel CPU analysis yielded predictable end-to-end runtimes; configuration caps allowed tuning of throughput versus latency.

CHAPTER 6 CONCLUSION AND FUTURE SCOPE

The system delivers a trustworthy offline baseline for BM@N detector health: interpretable PCA-based anomaly detection, physics-informed validation, and BMNROOT-faithful visuals. Its architecture is production-oriented (robust error handling, deterministic artifacts) and extensible.

6.1 Prospects

- Strengthen parallel processing and batch analysis: safe multiprocessing for CPU-only stages, file batching/scheduling for throughput, and profiling-driven tuning.
- Expand physics rule sets and unit tests; calibrate thresholds per detector group.
- Software engineering hardening: CI/CD, extended test coverage, telemetry/observability, containerization, and versioned configuration bundles.
- Persist results to a database; expose FastAPI endpoints for run-to-run analytics and dashboards.
- Advanced ML roadmap: production-grade KernelPCA; representation learning with Autoencoders/VAEs; sequence models (e.g., LSTM/Transformers) for temporal patterns; and CNN-based computer-vision pipelines for detector image maps.
- Predictive maintenance & decision support: forecasting of detector health,

confidence-aware alerting, and human-in-the-loop policies for automated responses.

- Develop an online pipeline (ZMQ/WebSocket ingress), with back-pressure and an operator UI for real-time alerts and drill-downs.
- Plan microservices decomposition and evaluate GPU acceleration for model stages.

BIBLIOGRAPHY

[1] *BMNROOT*. Accessed 2025. BM@N Collaboration. 2025. URL: `https://git.jinr.ru/nica/bmnroot`.

[2] ROOT Team. *ROOT Data Analysis Framework*. Accessed 2025. 2024. URL: `https://root.cern/manual/`.

[3] ROOT Team. *PyROOT: Python Bindings for ROOT*. Accessed 2025. 2024. URL: `https://root.cern/manual/python/`.

[4] BM@N Collaboration. *BM@N Experiment at NICA - Official Collaboration Website*. `https://bmn.jinr.int/`. Accessed: September 18, 2025. Joint Institute for Nuclear Research (JINR), 2025.

[5] Donald H. Perkins. *Introduction to High Energy Physics*. 4th ed. Cambridge University Press, 2000. ISBN: 9780521621960.

[6] scikit-learn developers. *scikit-learn: Machine Learning in Python*. Accessed 2025. 2024. URL: `https://scikit-learn.org/stable/`.

[7] *FAIRRoot Framework*. Accessed 2025. GSI/FAIR. 2025. URL: `https://fairroot.gsi.de/`.

[8] Python Core Developers. *Python Global Interpreter Lock (GIL)*. Python docs and PEPs; Accessed 2025. 2024.

[9] ROOT Team. *ROOT RDataFrame: High-Level Interface for Data Analysis*. Accessed 2025. 2024. URL: `https://root.cern/doc/master/classROOT_1_1RDataFrame.html`.

[10] NumPy Developers. *NumPy Documentation*. Accessed 2025. 2024. URL: `https://numpy.org/doc/`.

[11] pandas Development Team. *pandas Documentation.* Accessed 2025. 2024. URL: `https://pandas.pydata.org/docs/`.

[12] Matplotlib Developers. *Matplotlib Documentation.* Accessed 2025. 2024. URL: `https://matplotlib.org/stable/`.