Nuclear Instruments and Methods in Physics Research A 1085 (2026) 171269

Contents lists available at ScienceDirect

Nuclear Inst. and Methods in Physics Research, A

journal homepage: www.elsevier.com/locate/nima

Technical notes

An online data analysis framework for small-scale physics experiments

H. Ramm [a,*], P. Simon [b], P. Alexaki [c,d], C. Arran [e], R. Bingham [f], A. Goillot [d], J.T. Gudmundsson [g,h], J.W.D. Halliday [f], B. Lloyd [a], E.E. Los [a], V. Stergiou [a,d], S. Zhang [a], G. Gregori [a], N. Charitonidis [d]

[a] Department of Physics, University of Oxford, Parks Road, Oxford, OX1 3PU, United Kingdom
[b] GSI Helmholtzzentrum fur Schwerionenforschung GmbH, Planckstrasse 1, 64291, Darmstadt, Germany
[c] Department of Physics, University of Liverpool, Brownlow Hill, Liverpool, L69 7ZX, United Kingdom
[d] European Organization for Nuclear Research (CERN), CH-1211, Geneva 23, Switzerland
[e] Department of Physics, Lancaster University, Lancaster, LA1 4YB, United Kingdom
[f] STFC, Rutherford Appleton Laboratory, Chilton, Didcot, OX11 0QX, United Kingdom
[g] Science Institute, University of Iceland, Dunhaga 3, Reykjavik, IS-107, Iceland
[h] Division of Space and Plasma Physics, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Teknikringen 31, Stockholm, SE-10044, Sweden

ARTICLE INFO

Keywords:
HiRadMat
Super proton synchrotron
Laboratory astrophysics
Accelerator physics
Data collection
Data analysis

ABSTRACT

A robust and flexible architecture capable of providing real-time analysis on diagnostic data is of crucial importance to physics experiments. In this paper, we present such an online framework, used in June 2025 as part of the HRMT-68 experiment, performed at the HiRadMat facility at CERN, using the Super Proton Synchrotron (SPS) beam line. HRMT-68 was a fixed-target laboratory astrophysics experiment aiming to identify plasma instabilities generated by a relativistic electron–positron beam during traversal of an argon plasma. This framework was essential for experimental data acquisition and analysis, and can be adapted for a broad range of similar-scale experiments with a variety of experimental diagnostics, even those without a standard direct network communication interface. The developed framework's customizable design enabled us to rapidly observe and extract emergent features from a diverse range of diagnostic data. Simultaneously, its modularity allowed for a quick introduction of new diagnostic devices and the modification of our analysis as features of interest were identified. As a result, we were able to effectively diagnose equipment malfunction, and infer the beam's response to varying bunch duration, beam intensity, and the plasma state without resorting to offline analysis, at which time adjustment or improvement would have been impossible. We present the features of this agile framework, whose codebase we have made publicly available so that it may be adapted for future experiments with minimal modification.

1. Introduction and motivation

Small-scale accelerator experiments often face constraints from limited beam time and resources. As a typical example, the experiments in the CERN's North Experimental area are allocated 1 week of total beam time, which includes setup, installation, beam tuning, data taking, and decommissioning [1]. Furthermore, experiments not undertaken at state-of-the-art facilities, for example the European Organization of Nuclear Research (CERN), will not have access to specialist frameworks, such as CERN's Accelerator Data Logging Service (NXCALS) [2]. Smaller experimental facilities which typically accommodate test-beam users and experiments, among them INFN-LNF [3] or NCSR Demokritos [4], lack the NXCALS infrastructure, increasing reliance on custom and dedicated equipment to perform online and offline data analysis. Even when such frameworks are present, they may be insufficiently flexible to address experimental needs. In novel platforms like the newly established FIREBALL platform at CERN [5], experimental demands often change drastically during runtime, and frameworks that are flexible and capable of addressing these changes are essential.

Though there are conceptually more sophisticated and more reliable frameworks in operation, such as CERN's JCOP/WinCC Open Architecture [6], we have chosen to instead deploy our framework during experimental runtime in order to maintain unlimited control over its functionality, code, and integration. This was essential for continuity between incarnations of the FIREBALL experimental platform, providing us with a framework tailored to our particular experimental require-

* Corresponding author.
E-mail address: hayden.ramm@physics.ox.ac.uk (H. Ramm).

https://doi.org/10.1016/j.nima.2025.171269
Received 25 November 2025; Received in revised form 28 December 2025; Accepted 29 December 2025
Available online 5 January 2026

ments, and allowing internal continuity and collaborative modification between subsequent iterations of the experiment. Additionally, our framework was relatively quick to construct, minimally complicated, and highly customizable, and we modified it with ease even during our allocated beamtime to address issues ranging from a camera's optical misalignment to equipment malfunction. Such malfunctions were common and often critical; our Czerny–Turner monochromator [7], tasked with detecting synchrotron emission from within the plasma, ceased operation during experimental runtime. Though the device registered as still operational, it failed to capture images when triggered, and the cause of this failure remains unknown. Additionally, due to damage to an optic fiber, we could not deploy one of our optical components as planned. The code's modular structure, wherein devices can be trivially added or removed from the data extraction protocol, allowed the framework to continue running unimpeded despite these types of failure. On our small collaboration, such obstacles are difficult to resolve due to the lack of technical expertise with respect to specialist equipment.

HRMT-68 differed from its sister experiments at CERN in that it collected a comparatively small amount of data, approximately 20 GB in total. This is in sharp contrast to LHC-centered experiments. For example, during the LHC Run 3, LHCb collected raw data at a rate of 4 TB/s [8]. Similarly, ATLAS' Trigger and Data Acquisition (TDAQ) systems will be capable of processing data at an input rate of 4 TB/s after the High Luminosity upgrades at the LHC (HL-LHC) [9]. Our far smaller scope of data acquisition affected our design considerations, motivating the use of Python, whose performance in our framework is modest and would not be scalable to larger collaborations' experimental needs.

Particularly in small- or medium-size experiments, the scope of data collection, and the often relatively small fraction of beam spills (or pulses) corresponding to events of interest, motivates the development of software frameworks not only capable of discerning events of interest among a moderate quantity of data, but also of performing analysis and extracting features of interest *during* experimental runtime. This ability to perform data analysis in real-time is of immense value in cases where offline analysis would fail to identify faults such equipment failure or beam misalignment until long after the experiment's conclusion.

In this technical note, we present such a robust, flexible, and reusable framework for online data analysis of experimental diagnostics, first employed during the HRMT-68 experiment performed at the HiRadMat facility [10] at CERN, part of the Super Proton Synchrotron (SPS) accelerator [11]. This was a laboratory astrophysics (see the discussion of recent advances in laboratory astrophysics by Savin et al. [12]) experiment, which aimed to study plasma instabilities [13,14] produced by an electron–positron pair beam [15] traversing an argon plasma [5]. Our developed framework places an emphasis on readability and modularity, making it easily applicable and reusable for a diverse range of future experiments. Authored in Python, with the option to incorporate a JavaScript Object Notation (JSON) configuration file (see Section 4), it conforms rigidly to the object-oriented programming (OOP) paradigm for maximized encapsulation and abstraction [16,17], producing a software package that can be easily utilized by those with minimal programming expertise. The resulting framework executes most of its complex data extraction and analysis pipeline without directly interfacing with the user; this "under-the-hood" operation is one of the framework's primary strengths.

The present work is outlined as follows: the motivation and experimental layout of the HRMT-68 experiment is discussed in Section 2, provided to the reader as an exemplar application. We then present our methods of continuous data acquisition in Section 3, which was crucial in developing a central repository of raw data for online analysis during the experimental run. In Section 4, we discuss the code itself, outlining its hierarchical structure, modular design, and robustness, before quantifying its performance in Section 5. The paper concludes with a discussion of the framework's strengths and weaknesses, as well as any potential improvements for future incarnations of the code, in Section 6.

2. The HRMT-68 experiment

The HRMT-68 (FIREBALL-III) experiment utilizes a novel platform recently developed at CERN's HiRadMat facility, which generates an electron–positron pair-beam from the collision of a 440 GeV/c-momentum proton beam with a fixed-target, as detailed by Arrowsmith et al. [5,15,18]. This produced electron–positron pair-beam passes into a cylindrical argon plasma, produced via inductive discharge [18–20]. This plasma was contained in a glass plasma cell.

The complexity of this experiment required a diverse collection of diagnostic equipment, which, collectively, had to provide optical, spectroscopic, and magnetic field data from the plasma. A summary of the devices used, and their purpose, is provided in the form of Table 1. For details on the setup on previous incarnations of the experiment, we point the reader again to refs. [5,15,18].

Four screens made of Chromox, a chromium-doped alumina ceramic (99.5 % Al_2O_3, 0.5% Cr_2O_3) [18,21,22], were used in the setup. Chromox produces scintillation light in the visible spectrum at wavelengths of 691 nm and 694 nm [18]. Each of the four screens was imaged individually using four cameras. These four cameras, designated HRM$\{3,4,5,6\}$, were collectively dubbed the "Chromox cameras". Cameras HRM$\{3,4\}$ imaged the plasma cell directly, and were referred to as the "plasma cell" cameras; the two other Chromox cameras imaged screens on the far side of a dipole magnet, and were dubbed the "spectrometer cameras". Deflection of particles in a dipole field is a function of their energy, and hence scintillation patterns produced by deflected electrons/positrons as they impinge these two latter screens provide a measure of the electron/positron energy spectrum. Two screens were necessary, as electrons and positrons are deflected in opposite directions by a dipole magnetic field.

For any initial velocity, charged particles emit optical transition radiation (OTR) when passing between two media of differing dielectric constant; the variation in electric and magnetic fields across this boundary results in this radiative emission [23,24]. The collection of photons produced via OTR is often used in beam diagnostics to characterize the longitudinal density profile of accelerator beams [25,26], as was done in the HRMT-68 run. OTR was produced by introducing a copper foil into the path of the beam, with the subsequent emitted radiation collected by a "streak unit" [27] with an adjustable slit width. The information can be encoded in a 2D image, which contains information about the pulse duration of incident radiation, and is captured by a digital camera; such a digital camera was mounted on the streak unit during the experiment.

Synchrotron radiation is produced by the deflection of charged particles in magnetic fields [28]. In the HRMT-68 experiment, synchrotron emission was used to infer the interaction of the electron–positron beam with the magnetic fields produced in the plasma. Synchrotron radiation emission from the plasma was collected using a charge coupled device (CCD) [29] camera (Andor iStar [30]) mounted on a Czerny–Turner monochromator [7].

Magnetic field diagnostics for the plasma were provided by three B-dot probes, named for the overdot denoting differentiation of the magnetic field with respect to time: $\dot{\mathbf{B}}$ [31]. These probes measure the magnetic field strength within the plasma via a practical application of Faraday's Law: electromotive force (e.m.f.) induced in the coils of the B-dot probe is proportional to the rate of change of magnetic flux through the coils, using Eq. (1) (see Refs. [32]).

$$\varepsilon = -\frac{\mathrm{d}\phi}{\mathrm{d}t} = -N\frac{\mathrm{d}}{\mathrm{d}t}\int_S \vec{dS}\cdot\vec{B} = -N\int_S \vec{dS}\cdot\dot{\vec{B}} \tag{1}$$

The e.m.f. induced in the B-dot coils was measured using two identical four-channel oscilloscopes (Tektronix 6 Series MSO [33]).

Table 1
Summary table of devices present in the HRMT-68 experiment. Care is made to distinguish the spectrometer and streak unit from the two cameras (one per unit) mounted on them; raw data collected from the experiment was from these cameras, rather than the spectrometer and streak unit themselves.

Device		Purpose
Chromox Cameras	Plasma Cell Cameras	Observe transverse profile of electron–positron pair-beam.
	Spectrometer Cameras	Observe energy spectrum of electrons & positrons exiting plasma cell.
OTR Streak Unit + Camera		Produce spatially- and temporally-resolved information about OTR produced by pair-beam.
Synchrotron Emission Spectrometer + Camera		Resolve wavelengths of radiation present in synchrotron emission due to electron–positron beam's deflection by plasma **B**-field.
B-dot Probes + Oscilloscopes		Observe/plot **B**-field induced in plasma.

3. Data collection

Collection and logging of device data was potentially the most challenging aspect of the online framework. Data from each of the experiment's diagnostics (enumerated in Section 2) was collected into a centralized repository, with the online analysis framework constructed to accommodate this and read each device's data from a specified location (see Section 4).

3.1. Construction of a centralized data repository

CERN's sophisticated Accelerator Data Logging Service (NXCALS) [2] necessitates the existence of a special front-end, designated "Front-End Software Architecture" (FESA).[1] FESA was developed for LHC and its injectors, and is extremely versatile and adaptable. However, its implementation requires specialist expertise usually not present on small-scale international collaborations, and many experiment-specific pieces of equipment are not trivially compatible with FESA. For example, the HRMT-68 used a NAVIO matching network [34] for impedance matching when delivering RF power to the plasma cell; integrating the matching network into FESA would have required time and expertise that was unavailable to the collaboration.

Finally, it is generally known that the development of these generalized software frameworks (e.g. NXCALS/FESA) is driven by the global needs of each organization and its flagship experiments, leaving little to no space (in terms of resources or support) for the requirements and the versatility necessary for smaller experiments to be taken into account. This was our primary motivation for this work.

The variety of data from a range of diagnostics (Table 1) motivated the construction of an easily-accessible, experiment-specific central repository of data collected during the HRMT-68 campaign. Similar small-scale experiments may only have access to a generic cloud-based storage service accessible over a network connection, as was the case on HRMT-68. Therefore, our framework was designed to cater to this limitation.

In our case, we profited from CERN's online Jupyter environment, Service for Web-based ANalysis (SWAN, described in the CERN SWAN documentation [35]), which allows users to run code from any location. The Jupyter environment provides a user-friendly front-end for executing Python without the need for SSH connections to remote servers or installation of local versions of Python. SWAN is directly integrated with CERN's EOS Open Storage (EOS)[2] /CERNbox online file storage configuration; EOS is CERN's disk storage system, with a storage capacity of ∼780 PB [36], and "CERNbox" is CERN's cloud-based file-sharing and storage service [37]. Therefore, the use of SWAN permitted any user with EOS/CERNbox read and write permissions and an internet connection to run our software from any location.

Data from each diagnostic was assigned a unique directory in this central data repository, which was pointed to by the code's configuration (as described in Section 4). However, this framework would work equally well with any cloud-based or local file storage service.

In order to synchronize the different device data to the repository, each device was configured to either save data directly into CERNbox via a mounted network drive, or to a local directory synced continuously with the appropriate device directory on CERNbox, depending on the device. The speed of the synchronization process varied greatly between devices, from $\lesssim$5 seconds for devices with access to a mounted network drive, to $O(10)$ seconds for devices whose local directories were synced remotely.

3.1.1. Challenges with timestamp tagging

A central goal of the framework was the creation of a *consistent* scheme for logging all data acquired during the experiment. During the HRMT-68 experiment, several experimental conditions were continually modified. These included power supplied to the plasma cell, position of the graphite target used to produce the electron–positron beam, as well as the beam density requested from the SPS. Keeping track of which data files corresponded to which experimental conditions was therefore essential in subsequent data analysis. Therefore, we aimed to maintain a centralized "shot log" which tracked all experimental conditions during each beam extraction, and the time of each extraction. The aim was to then compare these extraction times to the time(s) at which each device was triggered.

Initially, the process of sorting devices' data via timestamp was to be carried out by automatically generating duplicates of the devices' output data files and appending a UNIX timestamp [38] to the filename. This proposed process is outlined in Fig. 1. The timestamp used to label device data was to be provided by CERN's Network Time Protocol (NTP) [39], and corresponds to the timestamp used to index CERN's accelerator cycles. This timestamp is termed the "cyclestamp". During our data collection and logging procedures, when relying on a connection the CERN NTP server, we observed up to an $O(10\ \text{s})$ discrepancy between the timestamps associated with each devices' data for the same extraction. This limitation was due to delays in the devices' data being saved to our cloud-based repository. However, this observed precision was more than sufficient to remove ambiguity surrounding which data corresponded to which extraction, as intervals between subsequent extractions from the beamline during the experimental run were around 90 s.

However, this initial method of tagging the filename with a timestamp directly proved infeasible for several reasons: firstly, it necessitates duplication of all data, resulting in a doubling of storage requirements. Given that hard disk space is often at a premium in small-scale experiments, this complication is unsatisfactory and will often be unacceptable. Secondly, timestamping the files (by creating a copies of each file with a modified filename) relied on continuous execution of a Python script which maintains a connection to the CERN NTP server via a WebSocket connection [40], which increased the framework's vulnerability to endemic network latency and connectivity concerns.

[1] See https://confluence.cern.ch/spaces/viewspace.action?key=FESA3.
[2] EOS acronym is recursive.

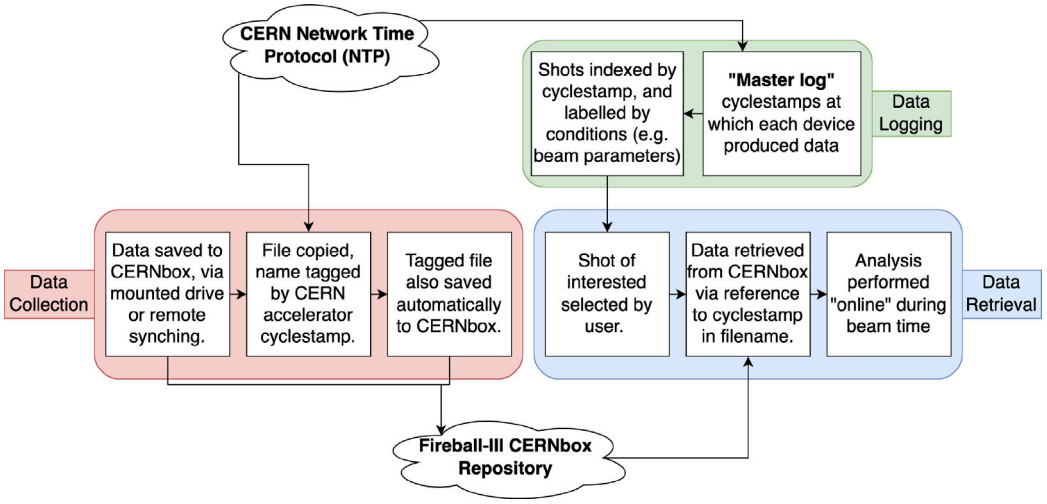

Fig. 1. Schematic diagram showing the **ideal** (and initially-proposed) data tagging, logging, and retrieval processes. In reality, the shot log could not be automatically generated, and the process of time-stamping data using the CERN accelerator cyclestamp proved infeasible due to instabilities in network connection.

As a result, only a fraction of collected data was correctly timestamped. We retain a description of this idealized process in order to motivate future efforts to dynamically tag data in such an unambiguous fashion while addressing storage and connectivity concerns.

A revised scheme (shown in Fig. 2) was introduced to compensate for the difficulties that arose while attempting to implement the scheme shown in Fig. 1. In place of an automated log of timestamps at which each device triggered, a simplified "secondary shot log" was maintained, which indexed the data from each device by the UNIX timestamp corresponding to the upload time on CERNbox rather than by relying on the cyclestamp. Unlike the central "master" log, which had to be maintained by hand and contained details of experimental conditions, the secondary log contained no detail of experimental conditions, such as target placement or conditions of the plasma. However, the secondary shot log was automatically generated, and hence resilient to human error.

This process, while cumbersome, was internally consistent and resulted in the creation of two centralized shot-logs for all diagnostic devices, where data files were indexed by a universal timestamp key. This eliminated ambiguity as to which data files corresponded to which experimental conditions and which beam extraction. These timestamp keys can then be fed into the framework's frontend to filter for data of interest across all devices.

Our framework retains the ability to access data for either method, both the case in which the filename is explicitly modified and appended with the UNIX timestamp, or the case in which data is logged by the time at which it first appeared on the EOS/CERNbox file system. The former case is preferable, as it provides all data with a permanent timestamp which is resilient to file modification or migration of data off the EOS/CERNbox storage system; both cases will result in changes to the files' modification time metadata, which was used in our revised method (Fig. 2) to index data.

4. Code structure and design

Having resolved the issue of indexing data by the time of acquisition, allowing us to filter out shots of interest, the subsequent goal of this project was to construct a set of analysis methods to be performed on the output data across all devices. Our desire to do so "online" meant that these methods had to be executable as soon as the data was available on our centralized CERNBOX/EOS repository. This demanded both flexibility (the ability to introduce and remove diagnostics at will) and specificity (with each device requiring its own unique analysis and processing). The codebase was written in Python 3.11.9.

The choice of Python resulted in somewhat of a compromise on speed and efficiency. However, it provided equally significant gains in flexibility. Python is by some metrics the most widely-used programming language [41]; hence our framework necessitates no specialist skills other than basic proficiency in a common programming language, compared to the specialist expertise required for device integration into NXCALS/FESA. This guarantees a baseline level of longevity (Python is unlikely to exit use in academic circles in the near future) and accessibility. Moreover, Python offers an extensive array of packages specialized in data extraction and analysis, such as the popular Pandas package [42] and the Numpy library, whose performance is optimized via the use of an underlying C/C++ backend for accelerated computation [43].

The code's hierarchical structure, shown schematically in Fig. 3, was chosen to allow a high degree of modularity, addressing both the issue of flexibility and the need to specify analysis individually for each device. Its high degree of modularity and class-based structure also allowed members of the collaboration not directly involved in the code's development to sample code piecewise from the analysis pipeline to perform their own specialized analysis in parallel with the online data collection.

4.1. Configurability

The code was designed to read commands from an input configuration (either via JSON file or a manually-edited "input configuration" dictionary in the Python environment) to maximize readability and customization. The JSON format is simple, has minimal syntax, and closely resembles plain English. The purpose of the configuration file was to reduce the proliferation of hard-coded values in the framework which would need to be modified between runs, a process prone to human error. In the input configuration, users could specify shot numbers of interest and/or request statistical averaging across these shots. As depicted in Fig. 4, the user is able to specify "background" data to be subtracted from data of interest. The configuration allowed for displaying images without performing any analysis to expedite execution speed; quantitative assessment of how this contributed to the code's execution speed can be found in Table 3.

4.2. Modularity

The code's modular structure, depicted in Fig. 3, focusing on the image analysis functionality, was key to its adaptability. As experimental goals were modified, or regions of interest identified, quick

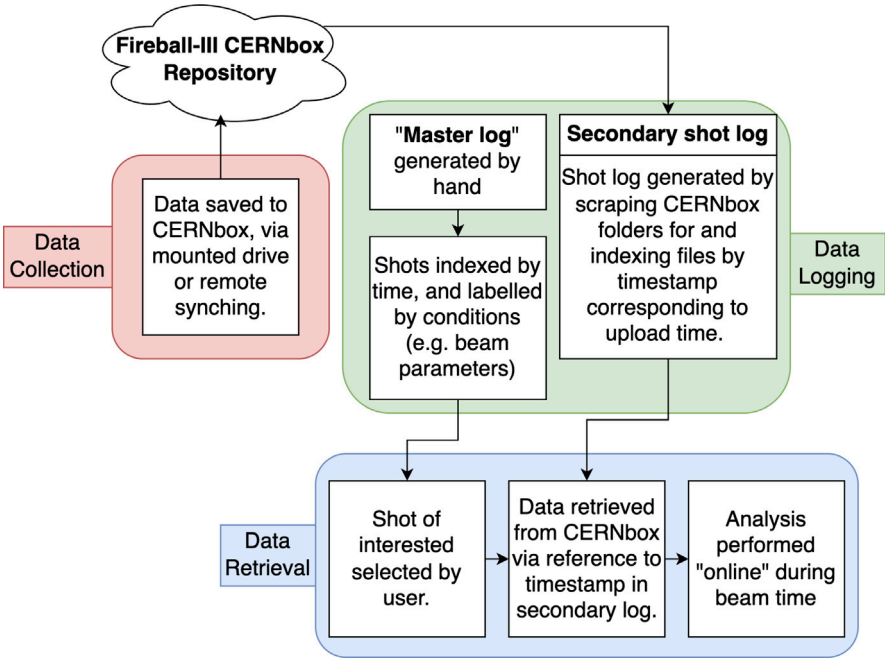

Fig. 2. Schematic diagram showing the system of data tagging, logging, and retrieval **in reality**, without tagging via cyclestamp. Here, a secondary log with minimal human-readable detail is introduced.

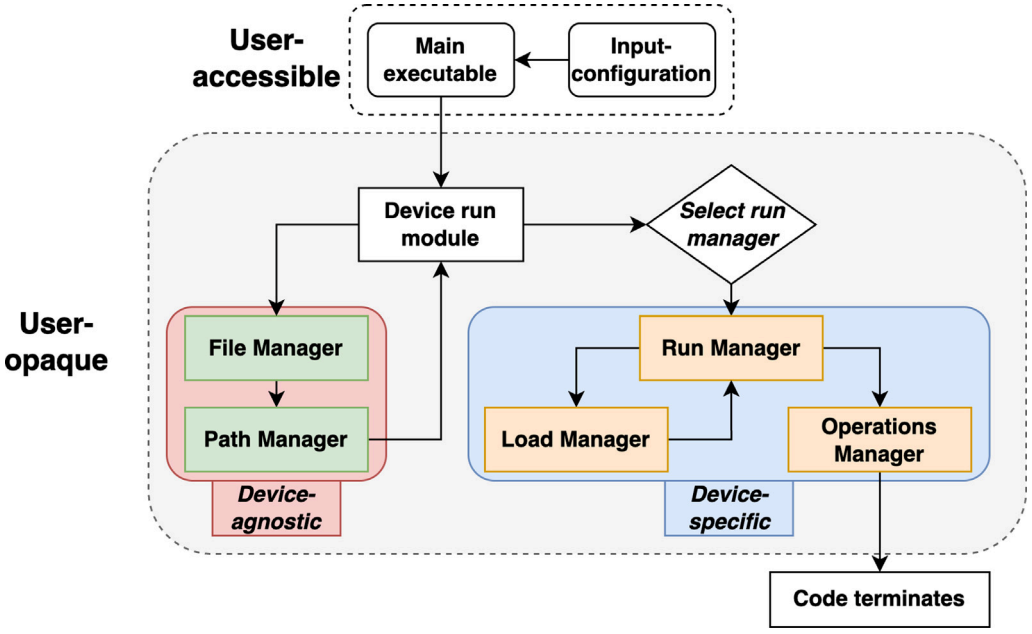

Fig. 3. Schematic tree of the framework's hierarchical structure, with modules shown in square boxes. Care is taken to distinguish device-agnostic modules (responsible for collecting file names from directories) from device-specific ones, where derivative classes for different device species were implemented to account for differing file formats and extensions, number of channels of data, and different plotting/analysis requirements for each device.

refactors could be made to analysis methods without disrupting the data collection and extraction processes. This allowed for a "plug-in-and-play" model: collaborators external to the project could implement their own analysis code, done independently and without any knowledge of the HRMT-68 online analysis framework, by simply inserting their extraction and analysis methods into the relevant modules in the framework. A detailed description of the different key-modules are described below.

The **file manager** module is tasked with creating an index of file *names* and their corresponding upload time, and this index is converted into a time-ordered index of file *paths* by the **path manager**, as described in Section 3. The **run** and **load manager** modules have two subclasses, one each for handling time-domain waveform data from oscilloscopes, and for handling bitmap image data from cameras. All device-specific analysis during runtime is controlled by the **run manager**, including statistical averaging and error propagation. The **load manager** subclass that the code calls is device-specific to account for the different file types and data logging structure of each device. All analysis and data visualization after the relevant data is extracted and preprocessed is handled by the **operations manager**, after which the code terminates.

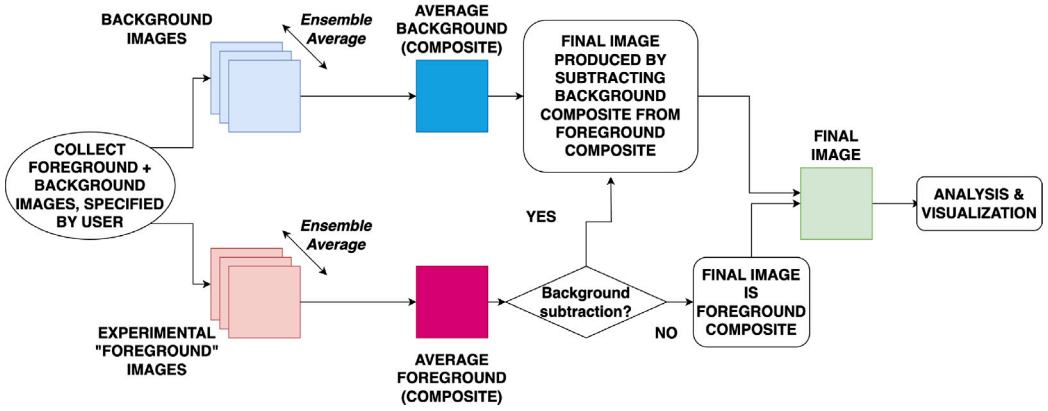

Fig. 4. Schematic showing the code's image averaging pipeline, including how multiple background shots could be combined into a single "composite background". Multiple foreground shots could also be combined into a "composite foreground" shot. Subsequent analysis is performed on the output of the subtraction between the composite images.

Table 2
Dimensions of images, in pixel number, produced by each of the three devices whose data was used to test the framework's performance.

Device	Image pixel dimension	Approx. size per image /MB	No. of shots collected
Synchrotron Spectrometer	1024 × 1026	4.2	≈ 100
OTR Streak Camera	2049 × 2049	14.5	≈ 300
Chromox Camera	349 × 605	5.0	≈ 1000

This hierarchical substructure is ideal as it allows for the straightforward, "plug-in" implementation of analysis methods for additional diagnostic devices, as required. As new devices are introduced to an experimental setup, desired analysis methods, such as visualization, cropping, and marginalization over an axis, can be implemented in the **operations manager** module. This can be done without modification to the **path**, **run**, and **load** manager classes, which contain generic functionality for logging and extracting data from any device.

5. Performance and results

In this section, we quantify the code's performance when producing analysis and/or visualization of diagnostic device data. For these tests, we tested the code with data from: the OTR streak unit camera, synchrotron spectrometer camera, a single Chromox camera (a single plasma cell camera, see Section 2), and an oscilloscope reading output from one of our B-Dot probes (see Table 1).

In Section 5.1, we present a summary of the code's performance in Tables 3 and 4, where we measure execution time and peak memory usage, respectively. The tests were performed over a remote connection to a CPU (Intel(R) Xeon(R) Silver 4216 CPU, 2.10 GHz [44]) on the CERN cluster, running an AlmaLinux9 OS distribution [45], via CERN's SWAN service, with a cumulative 8 GB of RAM and 2 cores allocated to the session. The ethernet connection of the remote CPU was configured to a ceiling of 10000 Mbps, and measurement of computer's download speed yielded ∼200 MB/s.

All tests were performed with the background subtraction routine to maximize computational complexity. This was produce estimates for a "worst-case" run time during testing. Background subtraction was done by averaging over 5 shots designated as "background" to create a "composite background" image, which was then subtracted from the foreground image (see Fig. 4). Summaries of pixel dimensions for the three types of device producing visual data are provided in Table 2

Each entry of execution time in Table 3 is an average over 10 trials.

As mentioned previously in Section 4.1, the code could have its configuration modified between runs. Users could specify that the framework perform no analysis and simply plot device data in order to increase execution speed in cases where it was only necessarily to visualize the camera images or oscilloscope traces, and we include such test cases in our performance evaluation. We provide a sample output from the analysis framework for one of the Chromox cameras (see Section 2), HRM3, in Fig. 5.

Results for tests of peak memory usage in Table 4 were averaged across 10 runs of the code. This was measured using Python 3's inbuilt memory profiler/memory usage function.

In order to account for the significant reduction in the code's speed introduced by calling analysis methods on the Chromox cameras, additional tests to quantify two of the most computationally-expensive analysis methods involved in the Chromox calculations were performed. These computationally expensive methods included calculating the first and second moments of image pixel intensity (i.e. the centroid of pixel intensity, and standard deviations on this centroid in both dimensions), as well as coordinate transformations from (x, y) 2D Cartesian coordinates to (r, θ) plane polar coordinates. The latter method was performed in order to generate radial and azimuthal marginals. These are summarized in Table 5.

5.1. Results and discussion

From the tests of code speed displayed in Table 3, we conclude that the foreground-averaging in the absence of analysis methods introduces negligible additional overhead for the Chromox camera and B-dot probe data. In the former case, execution time for foreground-averaging only introduces a $\sim(24 \pm 12)$ % increase in execution time, whereas effects to execution time for displays on the B-dot probe oscilloscope execution time are statistically negligible, and indeed execution time actually decreased, changing by $\sim(-3 \pm 30)$ % when averaging was introduced. We conclude that the code's statistical averaging methods are not the limiting factor, and that instead variations in connection/download speed between clusters accessed via SWAN and the CERNbox service, affected by network load and user demand, are far more likely to be responsible.

Inclusion of analysis methods, however, did introduce significant overhead in the execution process; in the case of the Chromox camera images: this resulted in percentage increases of $\sim(540 \pm 40)$ % and $\sim(480 \pm 60)$ % in execution time for the "single shot" and "average shot" cases, respectively. The methods used to calculate first- and second-order moments are computationally-costly, iterating over all pixels in

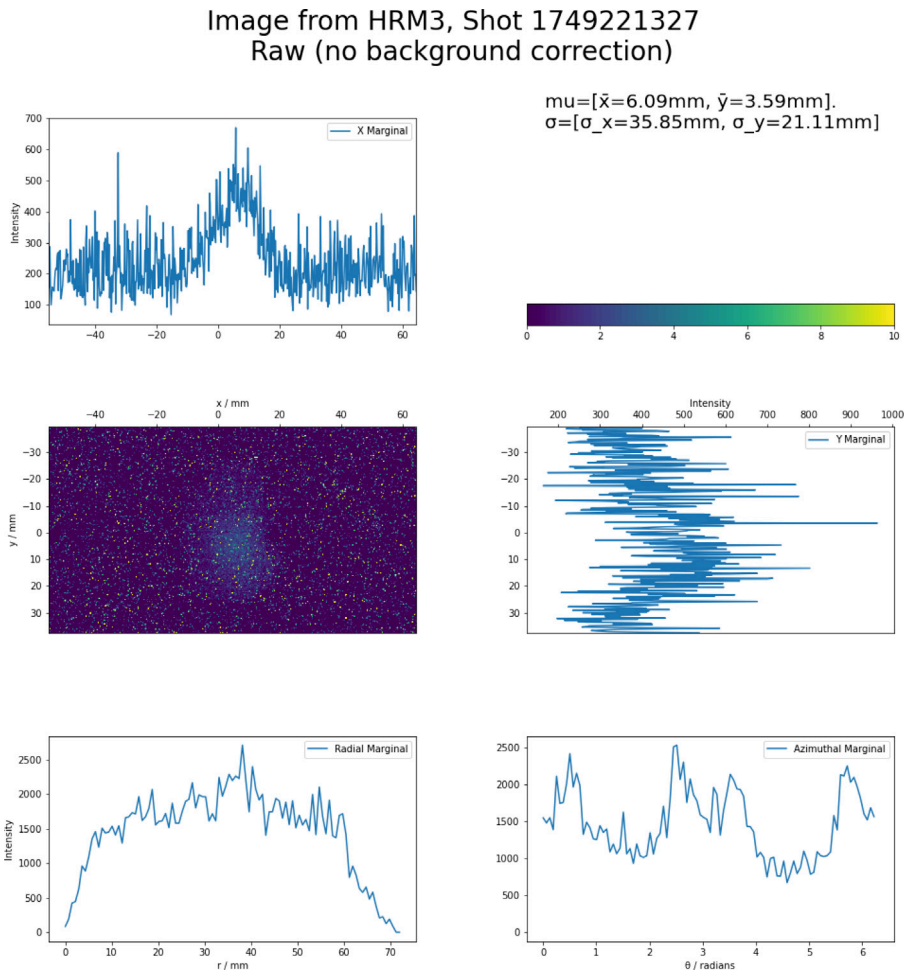

Fig. 5. Sample output from one of the Chromox cameras, HRM3 (see Section 2), tasked with monitoring the upstream cross-section of the plasma cell.

Table 3
Time-of-execution for the code for each species of device. Tests were done for cases with ("average shot") and without ("single shot") foreground averaging (see Fig. 4). For either case, trials were performed both with and without calling analysis on the resulting background-corrected data. "No-analysis" cases simply visualize the devices' data. Background subtraction was performed in all four cases, with a "composite background" image/oscilloscope trace produced by averaging five background images/traces. Values are statistical averages over 10 runs of the code.

Device	Code Execution/s			
	Single shot no analysis	Single shot with analysis	Average shot no analysis	Average shot with analysis
Chromox Camera	1.7 ± 0.1	10.8 ± 0.3	2.1 ± 0.2	12.1 ± 0.4
Synchrotron Spectrometer Camera	6.7 ± 0.3	6.3 ± 0.6	9.5 ± 0.6	10.4 ± 0.5
OTR Streak Unit Camera	24.8 ± 0.6	22.8 ± 0.4	37.9 ± 0.7	41 ± 2
B-dot Probe Oscilloscope	3.6 ± 0.9	3.9 ± 0.5	3.5 ± 0.5	5.0 ± 0.8

Table 4
Average peak memory usage in megabytes (MB) for each device in the four test scenarios, identical to those explored in Table 3. Values of peak memory usage are statistical averages over observed peak usage across 10 runs of the code.

Device	Average Peak Memory Usage/MB			
	Single shot no analysis	Single shot with analysis	Average shot no analysis	Average shot with analysis
Chromox Camera	536.5 ± 0.1	463.1 ± 0.3	587.1 ± 0.2	618.8 ± 0.1
Synchrotron Spectrometer Camera	537 ± 1	470 ± 10	589 ± 2	630 ± 30
OTR Streak Unit Camera	918 ± 4	890 ± 20	976 ± 1	1030 ± 20
B-dot Probe Oscilloscope	536.6 ± 0.2	463.0 ± 0.2	587.8 ± 0.1	619.0 ± 0.2

the image and therefore scaling as N^2 for an image of pixel dimension N. Transforming the images' pixels from Cartesian to planar polar coordinates also shared this computational complexity. As shown in Table 5, the execution times for these two functions were investigated individually. Cumulatively, they accounted for (58 ± 2)% of the entire runtime for calling analysis on the Chromox images.

For the B-dot probe oscilloscope analysis, percentage increases for execution time with the introduction of analysis was ∼(8 ± 30) % in the "single shot" case, and ∼(43 ± 30) % in the "average shot" case. The large confidence intervals on these results is a reflection that run-to-run fluctuations in execution speed were a significant fraction of the mean execution speed.

Table 5
Execution times for a full run of the analysis code on Chromox images from the tested Chromox camera, as well as the Chromox camera moment and coordinate transformation methods. These results are averages over 10 timed trials of code execution for "single shot" mode on one of the Chromox cameras.

Average total execution time for chromox run/s	Average total execution time, moment calculation methods/s	Average total execution time, coordinate transformation methods/s
10.6 ± 0.3	2.8 ± 0.1	3.39 ± 0.05

Operations with the streak unit camera were clear outliers in both execution time (Table 3) and memory consumption (Table 4). This is likely due to the larger pixel dimensions of the images from the synchrotron spectrometer and OTR streak unit cameras (1024 × 1026 and 2049 × 2049 pixels per image, respectively) compared to the dimensions of the single Chromox camera which was used during code testing (349 × 605 pixels).

Despite the increase in execution speed introduced by the larger pixel count in the spectrometer and streak unit images, introducing analysis resulted in only marginal increases in execution time. For the spectrometer camera, introducing analysis resulted in minor variation in execution time ((-6 ± 10) % (i.e. a *decrease*) for the "single shot" case, and (9 ± 9) % increase for the "average shot" case).

For streak unit analysis, variation between cases with and without analysis was larger ((-8 ± 3) % variation for the "single shot" case (once again, a *decrease* in execution time), and (8 ± 6) % increase for the "average shot" case). It is worth noting that the comparatively large variation between cases with and without analysis for the "single shot" case is unlikely representative of inefficiencies in the code; execution time was shown to actually decrease when analysis was introduced, despite the fact that operations performed by the code without calling analysis are a subset of operations performed when analysis is requested. This is most likely due to variations in network conditions.

We would also like to highlight that, for all devices, the analysis methods incur only a small amount of memory cost; indeed, memory consumption during tests across all devices actually decreased between the "single shot" cases when analysis was introduced; owing to the fact that there was an increase in memory consumption after the addition of analysis for the "average shot" cases, and the fact that the drop in peak memory consumption occurred for all devices, we suspect that this may be due to changes in the Jupyter kernel state between tests, though the exact mechanism explaining this occurrence remains unknown. This may also explain the decrease in execution time when analysis was introduced in the "single shot" cases for the OTR streak unit camera and spectrometer cameras, which are difficult to analyze due to the increased number of operations performed in the analysis case.

5.2. Evaluation

Though our framework's performance is not state-of-the art, its ability to address limitations on data collection (as described in Section 3) and cater to a wide range of diagnostics (Section 2) while maintaining a high level of flexibility and customizability made it invaluable to the experimental runtime. Our decision to implement our analysis methods directly in the framework's codebase also served as an invaluable, internal "alpha test" phase of our analysis scripts. For example, we were able to identify regions and shots of interest in preparation for our extensive offline data analysis efforts; thus, such specialist frameworks have significant potential in providing small-scale experimental collaborations with a "running start" on their analysis for subsequent publication.

6. Conclusions and future work

In this work, we have presented an online framework which is flexible to varying demands from a user, and which is moderately fast even under comparatively adversarial circumstances. It is highly modular, allowing easy implementation of data extraction and analysis code for additional diagnostic devices. Though the code's performance during experimental runtime was largely successful, we would like to also highlight the framework's weaknesses, and how it could be improved and modified in the future.

6.1. Introducing caching

Improvements could be made to execution speed by locally caching data from shots of interest. Often, it was desirable to plot data from the same shot multiple times in order to, for example, crop to regions of interest or adjust image contrast for visibility. This required multiple code runs, with each run limited by the speed at which data could be acquired over a network connection. In future, it would be useful to introduce an optional "caching" feature, allowing users to store data from a desired shot locally over a short period, with the option of "clearing" the cache when this data was no longer required.

6.2. Tagging by cyclestamp

Experimental collaborations using the framework should aim to have some universally-recognized system of logging data by acquisition time. At CERN, as discussed in Section 3, a strong candidate would be the UNIX timestamp corresponding to each accelerator cycle ("cyclestamp"), as data filenames can be "stamped" by unique UNIX timestamp corresponding to beam extraction times. The framework codebase is easily adjusted to allow indexing by UNIX timestamp so long as it knows the position of the timestamp in the filename string. More sophisticated standards of synchronization are possible and preferable to the kind employed in this framework. For example, the White Rabbit project at CERN provides sub-nanosecond synchronization precision using the IEEE 1588-2008 Precision Time Protocol (PTP) [46]. Future experiments may wish to distinguish between multiple data acquisition events for the same beamline extraction if devices are expected to trigger more than once at the sub-second level.

However, in the construction and execution of our framework, bottlenecks in speed and performance emerged primarily due to delays in the process of saving files from the multitude of diagnostic devices (described in Section 2) to our cloud-based CERNBox/EOS storage system, and this delay was folded into our latency budget constraints. Our framework has no method for increasing the speed of caching diagnostic data to the CERNBox/EOS repository after beam extraction, which is a hardware performance concern rather than synchronization rate limitation. Modifying our framework to include sub-nanosecond synchronization schemes will fail to address this shortcoming. Indeed, during our experiment, timestamp resolution needed only to be lower than the 90 s interval between beamline extractions (see Section 3.1.1). Synchronization methods with precision on or below the order of 1 s are readily achievable using standard operating system methods for synchronization between devices connected over Wi-Fi, such as the Windows Time (W32Time) Service [47], which employs NTP and can achieve $O(1\ \text{s})$ synchronization if network latency does not exceed 100 ms between devices.

Further improvements to the framework should focus on increasing the understanding of the underlying hardware, particularly the precise rate at which each diagnostic device logs data to the CERNBox/EOS repository, and the latency between beam extraction and data caching which subsequently emerges for each device. Revising the framework

to give increased understanding and control over the underlying diagnostic hardware is utterly crucial in improving the system we have described in this technical note. Once again, these concerns cannot be addressed by simply improving the precision of the underlying synchronization protocol.

6.3. Avoiding intermediate cloud-based services

Though the CERN SWAN service allowed direct user access to the CERNbox service, it introduced several obstacles. SWAN has no native functionality for editing JSON file types. Furthermore, the Jupyter kernel introduces further overhead, and online code refactors in the framework modules are not recognized unless the kernel is restarted. Additionally, SWAN requires a stable connection to the CERN network, which is in-demand during working hours, introducing unpredictable latency. A possible future workaround for experiments could be saving data to a local computer, rather than cloud-based service, on which the framework has been installed.

CRediT authorship contribution statement

H. Ramm: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Data curation, Conceptualization. **P. Simon:** Supervision, Software, Resources, Methodology, Investigation, Data curation, Conceptualization. **P. Alexaki:** Investigation. **C. Arran:** Writing – review & editing, Supervision, Investigation. **R. Bingham:** Supervision, Resources. **A. Goillot:** Validation, Supervision, Resources, Project administration, Methodology, Investigation. **J.T. Gudmundsson:** Supervision, Resources, Project administration, Methodology, Investigation. **J.W.D. Halliday:** Writing – review & editing, Validation, Supervision, Resources, Methodology, Investigation, Formal analysis, Data curation. **B. Lloyd:** Software, Methodology, Investigation, Data curation, Conceptualization. **E.E. Los:** Writing – review & editing, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **V. Stergiou:** Resources, Investigation. **S. Zhang:** Methodology, Investigation. **G. Gregori:** Supervision, Project administration, Funding acquisition, Conceptualization. **N. Charitonidis:** Writing – review & editing, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Code availability

All code is available and maintained at the following public repository: https://github.com/padishah115/fireball_onlineTF.

Funding acknowledgments

This project has received funding from the European Union's Horizon Europe Research and Innovation programme under Grant Agreement No 101057511 (EURO-LABS) and by the UK Research and Innovation (UKRI) Frontier Research Guarantee under Grant No EP/Y035038/1 (JETLAB).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

References

[1] D. Banerjee, J. Bernhard, M. Brugger, N. Charitonidis, N. Doble, L. Gatignon, A. Gerbershagen, The North Experimental Area at the CERN super proton synchrotron, 2021, http://dx.doi.org/10.17181/CERN.GP3K.0S1Y, URL https://cds.cern.ch/record/2774716.

[2] M. Sobieszek, V. Baggiolini, R. Mucha, C. Roderick, P. Sowinski, J.P. Wozniak, Implementing high performance & highly reliable time series acquisition software for the cern-wide accelerator data logging service, in: V.R.W. Schaa, A. Götz, J. Venter, K. White, M. Robichon, V. Rowland (Eds.), 19th International Conference on Accelerator and Large Experimental Physics Control Systems, 9 – 13. October, 2023, Cape Town, South Africa, JACoW Publishing, Geneva, Switzerland, 2023, pp. 1494–1498.

[3] A. Balerna, M. Ferrario, F. Stellato, The INFN-LNF present and future accelerator-based light facilities, Eur. Phys. J. Plus 138 (2023) 37, http://dx.doi.org/10.1140/epjp/s13360-022-03611-9.

[4] National Centre for Scientific Research "Demokritos", About us – national centre for scientific research "Demokritos", 2025, https://www.demokritos.gr/about-us/. (Accessed 18 November 2025).

[5] C.D. Arrowsmith, N. Shukla, N. Charitonidis, R. Boni, H. Chen, T. Davenne, A. Dyson, D.H. Froula, J.T. Gudmundsson, B.T. Huffman, Y. Kadi, B. Reville, S. Richardson, S. Sarkar, J.L. Shaw, L.O. Silva, P. Simon, R.M.G.M. Trines, R. Bingham, G. Gregori, Generating ultra-dense pair beams using 400 GeV/c protons, Phys. Rev. Res. 3 (2) (2021) 023103, http://dx.doi.org/10.1103/PhysRevResearch.3.023103.

[6] GEM DCS Team, "JCOP framework for WinCC OA – overview", 2021, https://cmsgemdcs.web.cern.ch/devguide/overview-JCOP/. (Accessed 10 November 2025).

[7] M. Czerny, A.F. Turner, Über den Astigmatismus bei Spiegelspektrometern, Z. Phys. 61 (11–12) (1930) 792–797, http://dx.doi.org/10.1007/BF01340206.

[8] M. Fontana, Triggering TB/s of data: The LHCb perspective, in: Proceedings of the CHEP Conference, Krakow, Poland, 2025, Presented on behalf of LHCb, 19–25 October 2025.

[9] R. Kopeliansky, ATLAS trigger and data acquisition upgrades for the high luminosity LHC, in: 31st International Symposium on Lepton Photon Interactions At High Energies, July 17 – 21, 2023, Melbourne, Australia, 2023, https://cds.cern.ch/record/2871280/files/ATL-DAQ-PROC-2023-007.pdf.

[10] I. Efthymiopoulos, C. Hessler, H. Gaillard, D. Grenier, M. Meddahi, P. Trilhe, A. Pardons, C. Theis, N. Charitonidis, S. Evrard, H. Vincke, M. Lazzaroni, HiRadMat: A new irradiation facility for material testing at CERN, in: C. Petit-Jean-Genaz (Ed.), 2nd International Particle Accelerator Conference, San Sebastian, Spain, 4 – 9. September 2011, 2011, pp. 1665–1667, https://accelconf.web.cern.ch/IPAC2011/papers/TUPS058.pdf.

[11] R. Arnaldi, Future facilities: the CERN SPS, 2025, arXiv:2505.10286.

[12] D.W. Savin, N.S. Brickhouse, J.J. Cowan, R.P. Drake, S.R. Federman, G.J. Ferland, A. Frank, M.S. Gudipati, W.C. Haxton, E. Herbst, S. Profumo, F. Salama, L.M. Ziurys, E.G. Zweibel, The impact of recent advances in laboratory astrophysics on our understanding of the cosmos, Rep. Progr. Phys. 75 (3) (2012) 036901, http://dx.doi.org/10.1088/0034-4885/75/3/036901.

[13] R. Lee, M. Lampe, Electromagnetic instabilities, filamentation, and focusing of relativistic electron beams, Phys. Rev. Lett. 31 (1973) 1390–1393, http://dx.doi.org/10.1103/PhysRevLett.31.1390.

[14] A.A. Schekochihin, Lectures on kinetic theory and magnetohydrodynamics of plasmas, 2024, Lectures for the University of Oxford MMathPhys/MSc in Mathematical and Theoretical Physics course, unpublished draft.

[15] C.D. Arrowsmith, P. Simon, P.J. Bilbao, A.F.A. Bott, S. Burger, H. Chen, F.D. Cruz, T. Davenne, I. Efthymiopoulos, D.H. Froula, A.G. ans J. T. Gudmundsson, D. Haberberger, J.W.D. Halliday, T. Hodge, B.T. Huffman, S. Iaquinta, F. Miniati, B. Reville, S. Sarkar, A.A.S. amd L. O. Silva, R. Simpson, V. Stergiou, R.M.G.M. Trines, T. Vieu, N. Charitonidis, R. Bingham, G. Gregori, Laboratory realization of relativistic pair-plasma beams, Nat. Commun. 15 (2024) 5029, http://dx.doi.org/10.1038/s41467-024-49346-2.

[16] M. Lutz, Programming Python, fourth ed., O'Reilly, Sebastopol, California, 2011.

[17] D. Phillips, Python 3 Object-Oriented Programming, Packt Publishing Ltd, 2015.

[18] C.D. Arrowsmith, The Stability of Electron-Positron Jets in Laboratory Plasmas (Ph.D. thesis), University of Oxford, 2024.

[19] C.D. Arrowsmith, A. Dyson, J.T. Gudmundsson, R. Bingham, G. Gregori, Inductively-coupled plasma discharge for use in high-energy-density science experiments, J. Instrum. 18 (4) (2023) P04008, http://dx.doi.org/10.1088/1748-0221/18/04/P04008.

[20] M.A. Lieberman, A.J. Lichtenberg, Principles of Plasma Discharges and Materials Processing, third ed., John Wiley & Sons, Hoboken, New Jersey, 2025.

[21] S. Burger, M. Turner, B. Biskup, S. Mazzoni, G. Switzerland, Scintillation and OTR screen characterization with A 440 GeV/c proton beam in air at the CERN hiradmat facility, in: Proceedings of IBIC2016, Barcelona, Spain, ISBN: 9783954501779, 2016.

[22] K.J. McCarthy, J.G. López, F.M. Hernández, B. Zurro, A. Baciero, M.A. Respaldiza, The response of a chromium doped alumina screen to keV and MeV ions, J. Nucl. Mater. (ISSN: 00223115) 321 (2003) 78–83, http://dx.doi.org/10.1016/S0022-3115(03)00208-3.

[23] T.F. da Silva, Beam monitoring using Optical Transition Radiation (OTR), Tech. Rep., Laboratório do Acelerador Linear (LAL), Laboratório de Implantação Iônica (LIO), and Instituto de Física da Universidade de São Paulo, 2010.
[24] B. Gitter, Optical transition radiation, Tech. Rep. CAA-TECH-NOTE-internal-#24, UCLA Department of Physics Center For Advanced Accelerators, 1992.
[25] R. Chen, Z. Gong, J. Chen, X. Zhang, X. Zhu, H. Chen, X. Lin, Recent advances of transition radiation: Fundamentals and applications, Mater. Today Electron. (ISSN: 2772-9494) 3 (2023) 100025, http://dx.doi.org/10.1016/J.MTELEC.2023.100025.
[26] K. Fedorov, P. Karataev, Y. Saveliev, T. Pacey, A. Oleinik, M. Kuimova, A. Potylitsyn, Development of longitudinal beam profile monitor based on coherent transition radiation effect for CLARA accelerator, J. Instrum. 15 (06) (2020) C06008, http://dx.doi.org/10.1088/1748-0221/15/06/C06008.
[27] Hamamatsu, Guide to streak cameras, 2008, Manufacturer information leaflet. https://www.hamamatsu.com/content/dam/hamamatsu-photonics/sites/documents/99_SALES_LIBRARY/sys/SHSS0006E_STREAK.pdf. (Accessed 01 August 2025).
[28] R.P. Walker, Synchrotron Radiation, Tech. Rep., Sincrotrone Trieste, Italy, 1994.
[29] R. Hui, Photodetectors, in: Introduction to Fiber-Optic Communications, Academic Press, ISBN: 978-0-12-805345-4, 2020, pp. 125–154, http://dx.doi.org/10.1016/B978-0-12-805345-4.00004-4.
[30] Oxford Instruments, Andor istar CCD series, 2025, Manufacturer datasheet. (Accessed 17 June 2025).
[31] S. Bose, M. Kaur, K.K. Barada, J. Ghosh, P.K. Chattopadhyay, R. Pal, Understanding the working of a B-dot probe, Eur. J. Phys. (ISSN: 1361-6404) 40 (1) (2019) 015803, http://dx.doi.org/10.1088/1361-6404/aaee31.
[32] D.J. Griffiths, Introduction to Electrodynamics, fourth ed., Cambridge University Press, Cambridge, United Kingdom, 2017.
[33] Tektronix, Tektronix 6 series MSO datasheet, 2024, https://www.tek.com/en/datasheet/6-series-mso. (Accessed 17 June 2025).
[34] Advanced Energy Industries, Inc., Navio™ Digital Matching Network — Quick, Accurate, and Repeatable Digital Impedance Matching, Tech. Rep., Advanced Energy Industries, Inc., 2018, Product Data Sheet (ENG-Navio-231-02 4.18), URL https://www.advancedenergy.com/getmedia/981d25fa-5584-4a6f-abdc-9f32cd3b1550/navio-product-data-sheet.pdf.
[35] CERN, CERN swan documentation, 2025, https://swan.docs.cern.ch. (Accessed 23 June 2025).
[36] CERN, CERN EOS documentation, 2025, https://eos-docs.web.cern.ch/diopside/introduction/index.html. (Accessed 23 June 2025).
[37] CERN, CERNbox documentation, 2025, https://cern.service-now.com/service-portal?id=service_element&name=CERNBox-Service. (Accessed 23 June 2025).
[38] P. Louis, Time on unix, 2020.
[39] K. Balakrishnan, R. Dhanalakshmi, B.B. Sinha, R. Gopalakrishnan, Clock synchronization in industrial internet of things and potential works in precision time protocol: Review, challenges and future directions, Int. J. Cogn. Comput. Eng. (ISSN: 26663074) 4 (2023) 205–219, http://dx.doi.org/10.1016/j.ijcce.2023.06.001.
[40] A. Melnikov, I. Fette, The WebSocket protocol, 2011, http://dx.doi.org/10.17487/RFC6455, RFC 6455, URL https://www.rfc-editor.org/info/rfc6455.
[41] TIOBE Software BV, TIOBE index for November 2025, 2025, https://www.tiobe.com/tiobe-index/. (Accessed 10 November 2025).
[42] T. pandas development team, Pandas-dev/pandas: Pandas, 2020, http://dx.doi.org/10.5281/zenodo.3509134.
[43] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. van Kerkwijk, M. Brett, A. Haldane, J.F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T.E. Oliphant, Array programming with numpy, Nature 585 (7825) (2020) 357–362, http://dx.doi.org/10.1038/s41586-020-2649-2.
[44] Intel Xeon Silver 4216 Processor, 22M Cache, 2.10 GHz Specifications, Tech. Rep., Intel, 2025, Product specifications webpage. Accessed: 2025-06-25.
[45] J. Aboutboul, Almalinux 9 release announcement, 2022, AlmaLinux blog post, https://almalinux.org/blog/almalinux-9-now-available/. (Accessed 23 June 2025).
[46] M. Lipiński, T. Włostowski, J. Serrano, P. Alvarez, White rabbit: a PTP application for robust sub-nanosecond synchronization, in: 2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, 2011, pp. 25–30, http://dx.doi.org/10.1109/ISPCS.2011.6070148.
[47] Microsoft, Support boundary for high accuracy time, 2025, https://learn.microsoft.com/en-us/troubleshoot/windows-server/active-directory/support-boundary-high-accuracy-time. (Accessed 28 December 2025).