ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD083

OVERVIEW AND STATUS OF THE MACHINE LEARNING DATA PLATFORM PROJECT*

C. K. Allen†, C. McChesney, M. Davidsaver, M. Frauenheim, L. B. Dalesio
Osprey Distributed Control Systems, LLC, Ocean City, MD, USA

Abstract

Osprey DCS is developing the *Machine Learning Data Platform* (MLDP), a public-domain, open-source product to support data science, machine learning and artificial intelligence applications for large particle accelerator facilities. Funding for MLDP development was obtained from the United States Department of Energy (DOE) through a Small Business Innovative Research (SBIR) grant. The project Phase II activity was completed in July of 2025; we present an overview then summarize project activity, achievements, and status.

INTRODUCTION

Background

The project was formally started in July 2022 with an SBIR Phase I award from the United States DOE, Office of High Energy Physics [1]. After the 6-month Phase I effort Osprey DCS was awarded a Phase IIA grant for continued MLDP development. The Phase IIA duration was from July 2023 to July 2025. Reported here are a review of the research and development efforts and accomplishments for the full 2.5-year project duration (Phase I and II).

Product Description

The *Machine Learning Data Platform* (MLDP) is a product providing full-stack support for data science, Machine Learning, and Artificial Intelligence (ML/AI) applications at particle accelerator and large experimental physics facilities. It supports ML/AI applications from front-end, high-speed acquisition of heterogeneous, time-series data, through data archiving and management, to back-end analysis. The MLDP embodies a "machine-learning ready" platform for data analysis and ML/AI applications in diagnosis, modelling, control, and optimization of these facilities. It provides data scientists and applications a consistent, data-centric interface to archive data standardizing implementation and deployment of ML/AI algorithms to different operations configurations within the same facility, or between facilities. Being an open-source, public-domain project, the MLDP is intended for broadest possible impact by increasing accessibility and minimizing the required expertise for installation and operation.

The MLDP can also be deployed at user facilities for experimental data collection, archiving, and analysis. It is capable of acquisition and archiving of heterogeneous data from experimental equipment (e.g., images, arrays, structures, etc.) along with system hardware configurations (e.g., scalars, tables), control system process variables, and any metadata required for provenance. Thus, the MLDP can manage experimental data through its entire lifecycle, from acquisition and archiving, through analysis and investigation, to release and final publication.

Below is a succinct outline of the MLDP design, more complete descriptions are available in previous articles (e.g., see [2-4]). The *Data Platform* (see below) main repository site also contains substantial online documentation, including papers, reports, and presentations [5].

Referring to Fig. 1, the MLDP is composed of 3 components: 1) the *Aggregator* performing front-end data acquisition, correlation, and staging, 2) the *Data Platform* responsible for data archiving, management, and analysis, and 3) the *Web Application* providing remote archive interaction using a web browser. The Aggregator [6] is deployed within Experimental Physics and Industrial Control System (EPICS) [7]. The Data Platform is a standalone system featuring an online installation utility and simple deployment [8]; it provides the data science functionality post-acquisition. The Web Application loads into web browsers via a separate server, providing remote access and interaction with the MLDP archive without any programming or scripting requirements [9].

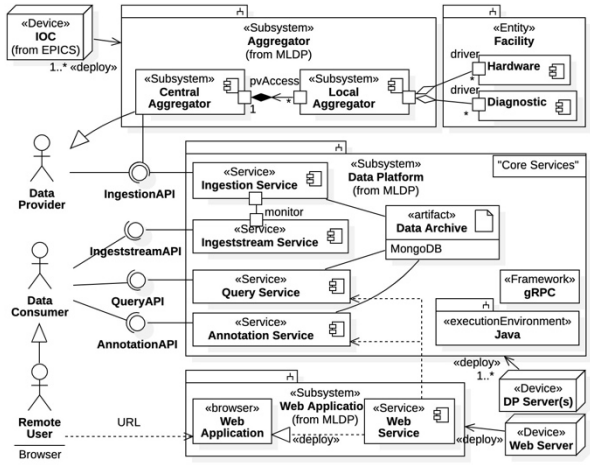

Figure 1: MLDP composite diagram.

Figure 1 provides a view the MLDP basic architecture, along with deployment and client relationships. The Aggregator system is distributed within the EPICS environment containing a *Central Aggregator* for coalescing and staging of acquired data. The Data Platform consists of multiple *Core Services* [10] interacting through an independent gRPC communication framework [11]. Each service has a gRPC Application Programming Interface (API) for interaction with clients, the archive, or each other. Data Platform *clients* are divided into *Data Providers* and *Data*

* Work supported by the U.S. Department of Energy, Office of High Energy Physics, SBIR Grant DE-SC0022583.
† email address allenck@ospreydcs.com.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD083

Consumers (these are simply definitions providing context within Data Platform). Data Providers are arbitrary sources of time-correlated, heterogeneous data (conforming to the ingestion API) contributing to the MLDP archive; note that the Aggregator is one such Data Provider. Data Consumers are any party interested in the MLDP archive, such as physicists, data scientists, and applications. Also seen in Fig. 1 is a Remote User interacting with the archive through the Web Application; it is a sub-class of Data Consumer.

RESEARCH AND DEVELOPMENT

Technical Objectives

The following is a list of the major technical objectives stated in the initial project scope:

- Performance: Data transport and archiving rates throughout the MLDP were to support, at a minimum, 4,000 scalar signals at a sample rate of 1 kHz. This translates to a data rate of 32 MBps.
- Heterogeneous Data Storage: Time-series data of differing types is to be stored and retrieved concurrently. This includes all common scalar types as well as complex types such as arrays, structures, and images.
- Archive Annotations: The MLDP was to support user annotation of archive data in the form of comments, data associations, and post-ingestion calculations.
- Installation and Deployment: The MLDP was to be deployable at existing particle accelerator and user facilities.
- Advanced Data Sciences: Phase II contained the addition of "Advanced Data Sciences" use cases, including online ingestion datastream access and processing for real-time applications.

The first three objectives were addressed immediately in the design phase. The remaining objectives were viable as project development continued.

Overview and Evolution

A MLDP prototype was built in Phase 1, completed at the end of 2022 [12]. It was originally intended for installation at facilities utilizing the EPICS system as the front end (Aggregator) was EPICS based. However, the archive management component of the prototype, then termed the *Datastore*, was built standalone and universally deployable. The Web Application was initially built as a development tool for inspection and verification of the data archive without formal API. However, it was found to have independent merit allowing anyone, developers and data scientists alike, remote access to the data archive without programming or scripting requirements or skills. Thus, there were 3 independent subcomponents of the prototype; this basic configuration remains into present day.

The MLDP prototype demonstrated proof of principle and a viable technical approach within the problem domain, however, it was not full featured and had serious issues regarding performance and reliability. In Phase I we exploited any available third-party systems and technologies to expedite prototype development. The approach allowed immediate evaluations identifying areas requiring future attention. Exhaustive evaluations of the prototype identified a comprehensive set of activities required to meet project technical objectives [12]. The areas of performance, reliability, and ease of use was a focus of Phase II.

The first operational Data Platform, Release 1.0, achieved an ingestion rate of ~100 MBps, providing a comfortable overhead of 3x over the target rate. The ingestion and archiving systems were again redesigned in Release 1.5 adding an additional factor 2 for maximum ingestion rates of 200 MBps. The accompanying Java API Library (JAL, see below) was then redesigned to allow multiple, concurrent data streams for ingestion, again increasing ingestion performance. Currently, the Data Platform has confirmed data rates of *over 500 MBps* over continuous, sustained ingestion yielding over 15x the Phase II goal. We expect another factor 2x increase in the latest release. The core code base has seen 3 major revisions now implementing a Common Core Service Framework (see below). New release versions are available on the Data Platform main repository site [8].

Phase II Technical Approach

The most promising approach to meet the above technical objectives was a complete redesign and rebuild of the MLDP archiving system (i.e., now the *Data Platform*). The reduction of 3rd party reliance was also a Phase II technical objective motivated by the open-source nature of the project, as well as performance improvement.

The initial Phase II effort involved technology evaluations in support of a robust archive system design [13], now termed the *Data Platform*. Design emphasis was on performance and commercial-grade code base. The Data Platform component is comprised of the following systems:

- Java [14] – The Java programming language offers a reasonable trade-off between performance, development effort, and ease of deployment. C++ provided the best performance but at a significant development increase and ecosystem requirements. Java also provides a platform independent deployment solution. Java project management is done with Maven [15].
- gRPC [16] – It was decided to maintain the archive system as a standalone, independent of EPICS. The gRPC system provides a fast, open-source, platform-independent, and language neutral communication framework that supports modularity and extensibility. The latter feature being particularly important for integration and Advanced Data Science efforts.
- MongoDB [17] – The MongoDB database system is now exclusive for all archiving, including time-series data, metadata, and archive annotations. Previously it was utilized solely for metadata due to its document-based storage structure. With redesign of the time-series data archive, the MongoDB system was found to have exceptional performance. Additionally, it is public domain and installs on most platforms.

The modern architecture of the MLDP is shown in Fig. 1. It is important to note that a new naming convention was adopted for the archive management system; *Datastore* was replaced by *Data Platform* (or DP). Since the

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD083

Datastore prototype and support was abandoned, it was prudent to identify it as such. The new convention clarifies the revised component from the prototype and identified the new code repositories. As indicated in the diagram the new Data Platform design is composed of collaborating services communicating through gRPC. The new design is performance-based, modular, and distributed. Additionally, the gRPC communications framework facilitates independence and the distributed, service-based nature. The first Data Platform implementation incorporating the design of Fig. 1 (Release 1.0) was completed late in 2023. It consisted of an operational Ingestion Service with testing utility; it supported only scalar data.

Many third-party systems and components were eliminated, including the InfluxDB [18] time-series database and the Spring and SpringBoot application frameworks [19]. Although designed for time-series data, InfluxDB functionality was not appropriate for heterogeneous data. The Spring and SpringBoot frameworks hindered performance goals and obfuscated operation. C++ gRPC provided exceptional performance, unfortunately, it is the most difficult to implement and deploy. Nonetheless, specialized MLDP components can be built in C++, particularly attractive for ingestion stream interaction and real-time processing. Thus, Data Platform C++ gRPC communications was well documented and available online [20].

Data Platform Communications Framework

As implied in Fig. 1, the Data Platform Communications Framework is gRPC based It is available in the Data Platform gRPC repository *dp-grpc* [11]. The external interfaces and data exchange messages are defined in Protocol Buffers format ("Protobuf") used to create high-level language interfaces to the gRPC library [21]. The *dp-grpc* repository is configured to automatically build and install the Java language implementation. Alternatively, Data Platform clients can download the repository and build the framework in the language of choice with the Protocol Buffers compiler "`protoc`". Figure 2 shows the Java implementation of the Communications Framework (`DP Communications`) for an arbitrary Data Platform service `DpService` defined in file `dpservice.proto`, discussed below.

High-Level Language Libraries

Not shown in Fig. 1 is the Java API Library (JAL) providing high-level programming language interaction with the Data Platform [22], isolating clients from the gRPC interface *dp-grpc*. Being Java based, the library was used extensively throughout project duration for testing of Data Platform APIs and services, as well as performance evaluations. The library contains a framework for building JAL applications and a suite of Data Platform testing applications; see online documentation for details [22]. A Python API Library (PAL) is intended for future development as many data science libraries are written in Python.

Service-Based Architecture

The Data Platform contains the *Core Services* of the MLDP. Shown in Fig. 1 are the *Ingestion Service*, the *Query Service*, the *Annotation Service*, and the *Ingestionstream Service* (abbreviated *Ingeststream*). Each service performs an essential function of the Data Platform and offers its client a well-defined API for interaction. Note in Fig. 1 that the Ingestion Service, Query Service, and Annotation Service have direct access to the MLDP data archive while the Ingestionstream Service connects to the Ingestion Service via the gRPC framework. The code base for the Core Services is in the *dp-service* repository [10].

The Ingestion Service provides the high-speed data ingestion and archiving functions with an interface based upon a heterogeneous "*data frame*." Any Data Provider conforming to the Ingestion Service API can supply data to the Data Platform, creating its standalone nature. The Ingestion Service also offers a subscribe/publish service to the live data stream, used by the Ingestionstream Service but also available directly to clients.

The Query Service is the archive interface seen by Data Consumers, such as data scientists, accelerator physicists, and applications programmers. For enhanced performance, query processing can be distributed between the Query Service and client. Specifically, as well as tabular queries, the Query Service offers streaming "raw" data queries where time-series data reassembly occurs on client platforms (e.g., the JAL library). This design off-loads processing from the Query Service increasing overall performance as the service can have multiple clients.

The archive annotations capability is managed by the Annotation Service with independent API. The Annotation Service also supports data exporting were archive data and annotations can be exported in various formats (e.g., CSV, HDF5, Excel). The export feature allows clients to simultaneously download time-series data along with calculation annotations for further local analysis and processing.

The Ingestionstream Service provides Data Consumers access to the ingestion datastream for real-time applications. Currently the Ingestionstream Service supports ingestion stream monitoring of *data events* where clients can specify a *data condition* triggering the event and request a data set within the response notification. For example, a client can request notification when a Process Variable exceeds a threshold (i.e., the trigger) then receive diagnostic data from the ingestion stream within a given time window.

As implied in Fig. 1 the new Data Platform architecture supports deployment on multiple host servers. For added performance deployment can utilize horizontal scaling of Data Platform where one or more Core Services can be deployed on dedicated servers. Likewise, the MongoDB database system can utilize multiple servers.

Core Service Common Framework

Each of Core Service shares the same design and implementation framework, the *Core Service Common Framework*. The framework is robust, efficient, and relatively straightforward. There was significant development effort here and we highlight the innovation.

Depicted in Fig. 2 is the base for an arbitrary Data Platform Core Service denoted `DpService` (e.g., the Ingestion Service, Query Service, Annotation Service, etc.). Figure 2 shows relationships between the gRPC library, the gRPC

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD083

Communications Framework [11], and the Core Service Common Framework [10]. Within the gRPC Communication Framework (box `DP Communications`) the file `dpservice.proto` defines the interface `DpService` and its required messages for data transmission. The interface `DpService` specifies several service operations labelled `operation1()`, `operation2()`, …, etc.

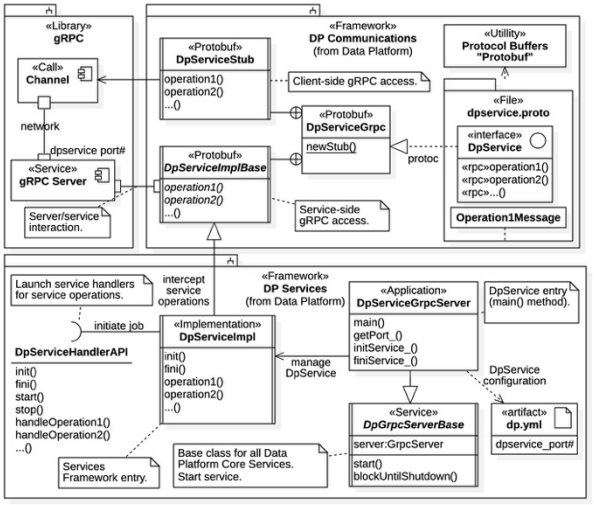

Figure 2: Core Service Common Framework base.

The Core Service Common Framework (box `DP Service`) provides two entry base classes, `DpServiceImpl` and `DpGrpcServerBase`, from which all services derive. The former is the Core Service entry point intercepting all gRPC traffic. The latter is the service application entry point containing the `main()` method by which all Java applications are launched. In our case the `DpService` application is implemented with derived class `DpServiceGrpcServer`; it starts the service on a gRPC server instance and performs startup and cleanup operations. It is also responsible for managing the service entry class `DpServiceImpl`, such as initialization, configuration, exception handling, and finalization operations.

It is the responsibility of any Core Service to implement a "service handler" performing all operations defined in the service interface (e.g., `operation1()`, `operation2()`, etc.). This is through delegation to specific "*service jobs*" defined for an operation. There are additional base classes and resources for building service handlers, jobs, and dispatchers. Each Core Service implements a service handler exposing a service handler interface, in our case `DpServiceHandlerAPI` seen in Fig. 2. The `DpServiceImpl` subclass relies upon the service handler for all `DpService` operations through this interface. For more details on the Core Service Common Framework see Phase II Year 1.5 Report [23]and Data Platform online documentation [5].

STATUS

Most of the Phase II project scope was addressed and are listed as technical achievements below. Some areas remain outstanding and are listed afterwards.

Technical Achievements

Most technical achievements were scoped within project requirements. However, others were pursued as opportunities arose, such as data exporting, Data Platform deployment and ease of use, and particularly, the outstanding performance now seen by the MLDP.

- Performance: Performance of the MLDP has greatly exceeded original objectives. Data rates for the Data Platform are confirmed at 500 MBps on development platforms, a factor 15x beyond original goals; we expect up to 1 GBps with recent improvements. Aggregator performance was also addressed with sampling rates now at 250 kHz in situ [24] at NASA's Artemis II test facility [25, 26], a factor 100x improvement.
- Communications Framework: Throughout the project, service interfaces saw repeated refactoring to accommodate new features, revised designs, and performance improvements. As of current Release 11.0 the Data Platform communications framework is relatively stable and support all features within Phase II scope except Algorithm Plugins (discussed below).
- Core Service Common Framework: We consider the Core Services Common Framework discussed above a technical achievement as it facilitated the Data Platform maintenance, robustness, upgradeability, and extreme performance characteristics.
- Data Archive: The MLDP robustly supports archiving of heterogeneous time-series data. Moreover, the MLDP maintains an archive of metadata for support of advance query building and data provenance.
- Archive Annotations: Post-ingestion annotation of time-series data is fully supported. Note that calculations obtained from archive data can be included alongside that data.
- Data Exporting: Clients can export archive data, metadata, and annotations in various formats (CVS, HDF5, Excel) for local analysis and processing.
- Ingestionstream Monitoring: The Ingestionstream Service provides access to the ingestion datastream in the form of subscribe/publish notifications to user-defined "data events". When a data condition is seen within the datastream, the client is notified and provided any requested data within the event definition.
- Installation and Deployment: The Data Platform has a formal installation and deployment system available at the main repository site [8]. Deployment consists of downloading an installer archive and unzipping into the install location. The installer contains scripts for starting and stopping services, monitoring service status, and utilities for benchmarking service performance on host platforms. Installation instructions and ecosystem requirements are available online.
- Data Archive Administration: Throughout the project tools for verification and validation of the Data Platform operations and performance were developed in conjunction with each service and ship within the installation [8]. These tools constitute the *Administration Library*. Although library components are

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD083

available, no formal Archive Administration tool has, of yet, been constructed (see below).

Outstanding Areas

Outstanding areas resulted primarily from unexpected effort required for archive annotations, data exporting, and the decision to target performance whenever presented.

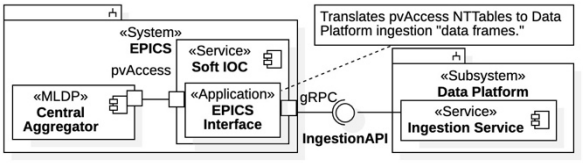

Figure 3: Data Platform EPICS interface.

- Full Integration: The Aggregator and Data Platform were developed independently and both subsystems have been tested and deployed independently. Currently the Aggregator stages data to HDF5 files which are then read into the Ingestion Service (e.g., see [27, 28]). Full integration of the Aggregator has yet to be implemented. A potential solution is shown in Fig. 3 where an *EPICS Interface* is deployed as an EPICS application [29], likely within a Soft IOC [30].
- Load Testing: Advanced load testing of the Data Platform has yet to be performed. The objective is to determine the maximum archive capacity of the Data Platform; that is, under what conditions it "breaks." Performance evaluations continued throughout the project but did not stress the archive capacity. An advanced Data Simulator capable of producing a variety of ingestion scenarios was recently completed. It can be configured to supply heterogeneous data from multiple, concurrent Data Providers at differing rates, different data types, executed over extended time scales. However, as of this writing testing has not begun.
- Algorithm Plugins: An intended feature of the Data Platform was an "algorithm plugin" framework. However, it was determined that such a framework was beyond the scope of low-level gRPC communications. The data event monitoring feature was successfully implemented in gRPC. It is our intent to implement an algorithm plugin framework in the JAL library.
- Python API Library (PAL): A Python language library was intended for Data Platform support as there are many public-domain Python data science libraries (e.g., see [31-34]). This activity was deferred until the Data Platform gRPC API was mature. Development of the PAL library is independent and self-contained; it can be built at any time, expediated from experience with JAL library implementation.
- Archive Administrator Tool: The intent is to create a client GUI application for the entire suite of archive administration applications and tools.

USE CASE: DATA CLEANING

To provide a cohesive application of the MLDP in data science operations we provide the example of "data cleaning." The example illustrates many of the features of the Data Platform Core Services in collaboration.

Figure 4 depicts the data-cleaning of an active ingestion stream. There are 3 actors, essentially independent processes: 1) a Data Provider actively supplying ingestion data, 2) a Data Monitor monitoring the ingestion stream for dubious data conditions (e.g., missing data, NaN values, "dark" signals, noise, etc.), and 3) a Data Cleaner responsible for data cleanup. The Data Provider supplies facility or experimental data to the MLDP archive in typical fashion. The Data Monitor is an independent online process that identifies anomalous ingestion data and annotates it, along with its condition, for later inspection and processing. The Data Cleaner is another independent process that locates the annotated data and "cleans" it according to condition and prescribed corrective actions. Potentially data cleaning could occur online using data event response data sets or offline using the Query Service. Note that the clean data is stored within the archive as calculation annotations associated with the original source data. Thus, users can inspect both the source data and the processed results.

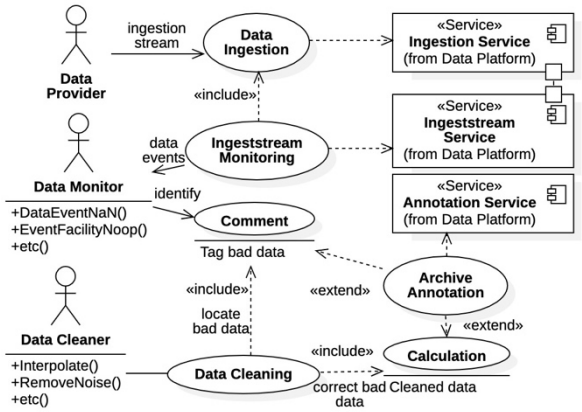

Figure 4: MLDP data cleaning use case.

Note that actors in the above use case are essentially facility dependent. Clearly the Data Provider is a property of the facility, typically data supplied by the control system from hardware, diagnostics, and experimental equipment. It may be possible to develop a general Data Monitor that inspects data for common anomalies, such as NaN, thresholds, and image corruption. Determination of corrective actions would likely be facility specific and thus the Data Cleaner application would be facility dependent. However, the MLDP supports all requirements of the use case.

CONCLUSION

The MLDP is operational, public-domain, and supports data science operations at large particle accelerator facilities. The Data Platform component managing the MLDP data archive is independent and can be downloaded and installed locally from the main repository site. Documentation for the Data Platform is exhaustive and available online. Most features within project scope were completed within the 2.5-year duration; some outstanding areas remain for future development as resources allow.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD083

REFERENCES

[1] Osprey DCS, "A Data Science and Machine Learning Platform Supporting Large Particle Accelerator Control and Diagnostics Applications", United States Department of Energy, Grant #DE-SC0022583, 2022.

[2] C. Allen, C. McChesney, M. Davidsaver, and L. Dalesio, "A data science and machine learning platform supporting large particle accelerator control and diagnostics applications", in *Proc. IPAC'24*, Nashville, TN, USA, May 2024, pp. 1843-1845. doi:10.18429/JACoW-IPAC2024-TUPS71

[3] C. McChesney, C. Allen, L. Dalesio, and M. Davidsaver, "The Data Platform: an independent system for management of heterogeneous, time-series data to enable data science applications", in *Proc. IPAC'24*, Nashville, TN, USA, May 2024, pp. 1839-1842. doi:10.18429/JACoW-IPAC2024-TUPS70

[4] C. McChesney, "A Data Science and Machine Learning Platform Supporting Large Particle Accelerator Control and Diagnostics Applications", presented at EPICS Collaboration Meeting, Oak Ridge, TN, USA, Sep. 2024, unpublished. https://github.com/osprey-dcs/data-platform/blob/main/doc/documents/presentations/old/202409-EPICS-meeting-dp-overview.pdf

[5] data-platform, https://github.com/osprey-dcs/data-platform/tree/main/doc/documents

[6] BSAS, https://github.com/mdavidsaver/bsas/tree/russo-vfield

[7] EPICS - Experimental Physics and Industrial Control System, https://epics-controls.org

[8] data-platform, https://github.com/osprey-dcs/data-platform

[9] dp-web-app, https://github.com/osprey-dcs/dp-web-app

[10] dp-service, https://github.com/osprey-dcs/dp-service

[11] dp-grpc, https://github.com/osprey-dcs/dp-grpc

[12] C. K. Allen, B. Dalesio, G. McIntyre, C. McChesney and M. Davidsaver, "Machine Learning Data Platform: Phase I Final Report", TM-01-2023, Osprey DCS, Ocean City, MD, 2023, https://github.com/osprey-dcs/data-platform/blob/main/doc/documents/reports/MLDP-Phase1-ver4.pdf

[13] Osprey DCS, "Phase II SBIR First Quarter Report - DE-SC0022583", Osprey DCS, Ocean City, Maryland, 2023, https://github.com/osprey-dcs/data-platform/blob/main/doc/documents/reports/DE-SC0022583-Q1-Report-ver1.pdf

[14] Java, https://www.java.com/en

[15] Apache Maven Project, https://maven.apache.org

[16] gRPC, https://grpc.io

[17] MongoDB, https://www.mongodb.com/home

[18] InfluxDB, https://www.influxdata.com

[19] Spring, https://spring.io

[20] Evaluation of C++ gRPC, https://github.com/osprey-dcs/data-platform/blob/main/doc/documents/reports/EvaluationOfCppGrpc-ver7.pdf

[21] Protocol Buffers, https://developers.google.com/protocol-buffers

[22] Java API Library (JAL), https://github.com/osprey-dcs/dp-jal

[23] DE-SC0022583 SBIR Initial Phase II, Performance Report: May to November 2024, https://github.com/osprey-dcs/data-platform/blob/main/doc/documents/reports/DE-SC0022583-Y1.5-Report-ver12.pdf

[24] Osprey Quartz DAQ-250-24, https://github.com/osprey-dcs/quartz-daq-250-24

[25] Artemis II, https://www.nasa.gov/mission/artemis-ii

[26] grc-atf-fdas, https://github.com/osprey-dcs/grc-atf-fdas

[27] Managing BSAS-NC Software and Data, https://confluence.slac.stanford.edu/spaces/ppareg/pages/610472556/Managing+BSAS-NC+Software+and+Data

[28] Beam Synchronous data Acquisition Service, https://github.com/slaclab/bsas

[29] Application Developer's Guide, https://docs.epics-controls.org/en/latest/appdevguide/AppDevGuide.html

[30] How to Set Up a Soft IOC Framework on Linux, https://epics-controls.org/resources-and-support/documents/howto-documents/soft-ioc-framework-linux

[31] Keras, https://keras.io

[32] TensorFlow, https://www.tensorflow.org

[33] PyTorch, https://pytorch.org

[34] scikit-learn, https://scikit-learn.org/stable/index.html