PROVIDING REPRODUCIBILITY AND ACCESSIBILITY OF RESEARCH SOFTWARE DEVELOPMENT FOR ACCELERATOR PHYSICS AT KARA

F. Donoso*, E. Bründermann, M. Frank, S. Funkner, J. Gethmann,
S. Masoumi, A.-S. Müller
Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract

Modern accelerator facilities generate heterogeneous data from diagnostics, sensors and simulations, making it difficult to manage, reproduce, and contextualize results as software and models evolve. While FAIR principles (Findable, Accessible, Interoperable, Reusable) are increasingly applied to research data, the iterative development of scientific software, with its rich metadata and benchmarks, rarely follows them. Small changes in compiler flags, dependencies, or hardware can alter outcomes, yet are often undocumented. We address this gap with BenchTune, a telemetry and reporting layer in C++ and Python that interfaces with the Kadi4Mat (Kadi) virtual research environment. BenchTune records metadata for each algorithm run, compiler information, parameters, metrics and stores it in Kadi as portable, traceable research artifacts. A project-view plugin in Kadi visualizes the evolution of a research project, linking code versions, datasets, and benchmark runs into coherent workflows.

As a result, our contribution provides a reproducible framework that supports FAIR principles for research data, enabling sustainable and collaborative research in accelerator physics and beyond.

MOTIVATION

The modern landscape of accelerator physics is increasingly defined by the intersection of high-performance hardware and sophisticated software stacks. At the Institute for Beam Physics and Technology (IBPT), our primary mission is accelerator research at the Karlsruhe Research Accelerator (KARA), using it as a test facility for studies with electron beams, beam diagnostics, and related developments such as magnet systems.

KARA serves as a critical testbed for novel diagnostics, beam dynamics studies, and advanced light source applications, generating large and heterogeneous streams of scientific and technical data even for a medium-sized accelerator. The accelerator infrastructure alone monitors around 30,000 data points across roughly 900 networked devices, with sensors and controllers operating at diverse update rates. The resulting data volume already reaches the order of 100 TB.

In accelerator research, workflows can increasingly be organized from data acquisition through analysis to publication. In our research environment, the provenance of raw and processed data is already treated as a central concern and is addressed according to established FAIR principles [1]. At IBPT, this process has recently been strengthened by the adoption of Kadi4Mat [2] (Kadi), a virtual research environment that supports the structured organization of data, metadata, and links between intermediate and final results within scientific workflows [3].

However, capturing complete algorithm provenance is significantly harder, yet essential for aligning with FAIR for Research Software [4–6] (FAIR4RS). This is particularly true for exploratory research software, which is often developed and executed on personal workstations, local servers, or temporary high-performance computing environments. Although final software versions may eventually be archived in long-term repositories such as Zenodo, the provenance of individual software executions is rarely preserved as a research object in its own right [7]. Such execution-level provenance extends beyond the source code and includes the scientific literature motivating the implementation, metadata describing the execution environment, code variables, runtime parameters, intermediate results, failures, performance measurements, and the configuration and build process of the software artifact. Consequently, important contextual information about how scientific results were actually produced is often lost.

This situation reveals a critical FAIR-gap [4–6] in software driven research. Consider a common scenario: a researcher produces a set of high-quality plots, pushes the code to a Git repository, and archives the dataset on Zenodo. Despite this, a scientist attempting to reproduce the results at a later stage may still fail. The code in the repository may have changed since the plots were originally produced, the exact runtime parameters may never have been documented, and the computation itself may have relied on a specific compiler version, build configuration, dependency stack, or hardware platform that is no longer accessible. Moreover, numerical libraries can exhibit different behavior depending on machine architecture, compilation flags, or backend implementations, leading to different outcomes even when the nominal algorithm remains unchanged. In this sense, access to source code and input data alone is insufficient to guarantee reproducibility, as the metadata describing the software execution, runtime environment, and configuration has not been properly persisted [8].

Software provenance is particularly important in accelerator physics, where many beam properties cannot be measured directly but must be reconstructed from heterogeneous diagnostic data. Many quantities of scientific interest are obtained only through reconstruction processes that combine time-resolved measurements, acquisition systems, physical models, and numerical analysis tools. Because the physics

* felipe.donoso@kit.edu

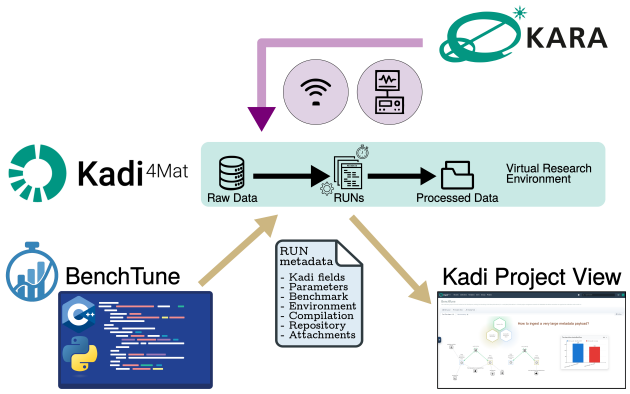

Figure 1: Overview of the KARA research stack, illustrating the pipeline from initial sensor data and metadata acquisition to the persistence of algorithm execution RUNs and their visualization via the Kadi Project View.

is reconstructed through code, the scientific software often evolves iteratively alongside the research questions. Simulation codes, reconstruction pipelines, optimization routines, and benchmarking scripts are continuously revised as models improve and new measurements become available. In practice, these intermediate development steps, together with their runtime context, are rarely preserved as first-class research objects. Consequently, the path from raw measurements to published scientific conclusions remains only partially documented, while the software itself remains insufficiently accessible, comparable, and reusable.

The work presented in this paper addresses this gap by extending provenance tracking beyond traditional data and metadata records to individual software execution instances, hereafter denoted as RUNs. In this notation, RUN refers to a research object representing one execution of research software. Our approach persists both the resulting data and the comprehensive execution metadata of each algorithm run directly within the Kadi environment. While Kadi is typically utilized to map direct, explicit relationships between data files, we extend this paradigm by embedding the execution RUN as an intermediate node in the provenance chain. Consequently, software execution metadata becomes an integral, traceable component of the documented research workflow. To support this approach, we introduce two key tools: *BenchTune*, which sends run metadata directly to Kadi, and *Kadi Project View*, a visual management extension for organizing and exploring these relationships.

By integrating these tools, we provide a framework that is intended to make research software at KARA as traceable and reproducible as the physical experiments themselves.

KARA RESEARCH STACK

During beamtime, KARA generates massive, continuous streams of operational data. This routine machine data and its associated operational metadata are automatically ingested and persisted in an Apache Cassandra [9] database, ensuring high throughput time-series logging. However, data acquired during dedicated research experiments often follows a more heterogeneous trajectory. Depending on the workflow, experimental data may be routed to the institute's central repository, stored within KIT's Large Scale Data Facility (LSDF), or temporarily saved on local researcher workstations.

In this heterogeneous environment, data and metadata are not automatically transferred to Kadi. Instead, researchers decide which artifacts should be persisted in the local Kadi instance, which is itself connected to the LSDF infrastructure. For experimental datasets, this archived information should include not only the acquired data itself, but also rich metadata describing the experiment and the state of the accelerator at the time of acquisition.

Once persisted, experimental data typically enters a second stage of processing and interpretation. This stage often involves additional measurements as well as custom research software developed by the researchers, primarily in Python and C++. Such software is used for analysis, reconstruction, simulation, and benchmarking, and therefore forms an essential part of the scientific workflow [8].

To capture the provenance of these computational steps, the metadata and outputs produced during individual RUNs of research software are transferred to Kadi through the BenchTune library. In this way, each run becomes a traceable research artifact that links raw data, processed data, runtime parameters, and execution context. As illustrated in Fig. 1, this extends the research stack beyond conventional data storage by introducing algorithm runs as explicit provenance objects within the workflow.

A key improvement enabled by this research stack is the ability to use this enriched and dynamic provenance not only for preservation, but also for inspection and interpretation of the research process itself [3]. Through the Kadi Project View plugin developed for Kadi, these relationships can be explored visually, allowing researchers to analyze, preserve, and communicate the evolution of a project in a more intuitive and structured way.

BENCHTUNE: RUNs TELEMETRY

BenchTune is a telemetry and reporting layer for C++ and Python that captures system metadata, structures runtime parameters, and serializes benchmarking results into portable formats. It supports several output backends, including console output, plain-text files, YAML, and JSON, thereby enabling both human-readable reports and machine-readable artifacts. The transmission of this metadata payload to Kadi is optional and can be configured independently of the local export format.

As a lightweight library, BenchTune can be integrated directly into existing C++ and Python codes with minimal effort. Once enabled, it automatically gathers metadata from the execution environment, including kernel, storage, CPU, compiler, virtualization, distribution, memory, executable information, and user-defined parameter values provided by the code itself. This allows each execution to be documented

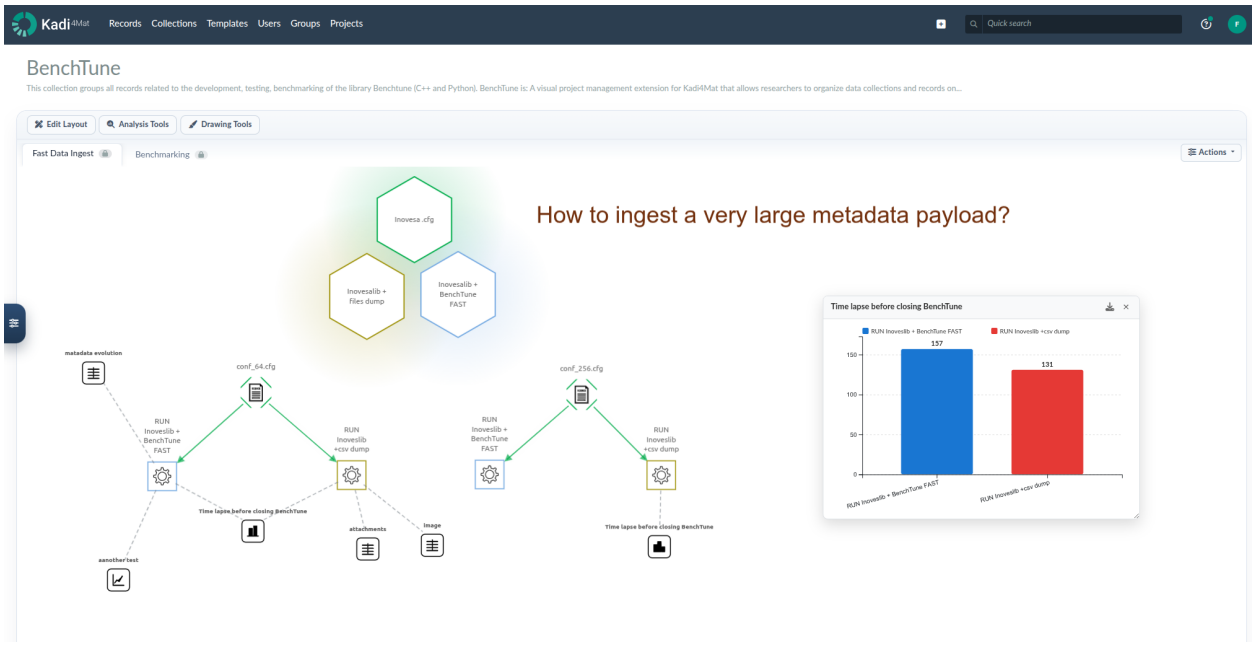

Figure 2: Kadi project view canvas example.

/vskip2mm

as a traceable run enriched with relevant computational context.

The following code snippet shows an example of BenchTune usage in a C++ program.

Listing 1: BenchTune C++ use example.

```
#include <BenchTuneManager.hpp>
using namespace BenchTune;

BenchTuneConfig cfg;
cfg.name = "RUN name";
cfg.output_directory = "/output/myRUN";
cfg.base_filename = "RUN_local_file";
cfg.description = "This is a RUN of my algorithm.";
// Metadata ingest configuration
cfg.metadata.all = True;
cfg.metadata.system = false;
// Kadi configuration
BenchTuneConfig::KadiOptions kadi;
kadi.enabled = true;
kadi.base_url = "kadi_url";
kadi.access_token = "kadi_token";
kadi.identifier = "record_id_kadi";
cfg.kadi = kadi;
auto benchtune = BenchTuneManager(cfg);
//Add metadata directly from code
benchtune->add("model/parameter", model->parameter);
benchtune->add("magic_number", 42);
// Write file and sent to kadi (if kadi available)
benchtune->close();
```

KADI PROJECT VIEW

Kadi Project View extends the Kadi environment with an interactive visual workspace in which researchers can organize records, collections, and their relationships on a two-dimensional canvas, as illustrated in Fig. 2. Rather than navigating research artifacts only through conventional lists and metadata forms, users can arrange project elements spatially to reflect the structure and progression of their work. This includes customizable visibility settings and flexible canvas interactions such as dragging, zooming, panning, and annotations. In this way, Kadi Project View provides a more intuitive representation of complex research projects, making the evolution of datasets, record versions, and their interconnections easier to inspect and communicate.

Beyond visualization, Kadi Project View also introduces analysis-oriented functionality directly into the project canvas. Records can be enriched with attached analysis tools that expose relevant metadata and derived results in an integrated and accessible manner. This allows users to inspect the history of artifacts, compare metadata across runs, and access associated outputs without leaving the project context. By combining visual workflow organization with embedded analysis capabilities, Kadi Project View transforms provenance from a passive record into an active tool for understanding, comparing, and communicating scientific work [3, 8].

DISCUSSION

This work shows that treating algorithm RUNs as first-class research artifacts within Kadi, through BenchTune and Kadi Project View, advances the implementation of FAIR4RS principles at KARA. To serve the broader scientific community, future work will deeply integrate the Project View into the Kadi ecosystem and expand BenchTune to support additional programming languages.

ACKNOWLEDGEMENTS

F.D. gratefully acknowledges support from the MathSEE PhD Bridge Program and the KIT Graduate School Computational and Data Science.

REFERENCES

[1] M. D. Wilkinson *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship", *Sci Data*, vol. 3, no. 1, p. 160018, Mar. 2016. doi:10.1038/sdata.2016.18

[2] N. Brandt *et al.*, "Kadi4Mat: A Research Data Infrastructure for Materials Science", *Data Science Journal*, vol. 20, p. 8, Feb. 2021. doi:10.5334/dsj-2021-008

[3] L. Griem *et al.*, "KadiStudio: FAIR Modelling of Scientific Research Processes", 2022. doi:10.5445/IR/1000151912

[4] A.-L. Lamprecht *et al.*, "Towards FAIR principles for research software", *Data Science*, vol. 3, no. 1, pp. 37–59, Jun. 2020. doi:10.3233/DS-190026

[5] N. P. Chue Hong *et al.*, "FAIR Principles for Research Software (FAIR4RS Principles)", 2021. doi:10.15497/RDA00068

[6] M. Barker *et al.*, "Introducing the FAIR Principles for research software", *Sci Data*, vol. 9, no. 1, p. 622, Oct. 2022. doi:10.1038/s41597-022-01710-x

[7] D. S. Katz, "Citation and Research Objects: Toward Active Research Objects", Jul. 2019. doi:10.5281/ZENODO.3338176

[8] S. R. Wilkinson *et al.*, "Applying the FAIR Principles to computational workflows", *Sci Data*, vol. 12, no. 1, p. 328, Feb. 2025. doi:10.1038/s41597-025-04451-9

[9] S. Marsching, "Scalable Archiving with the Cassandra Archiver for CSS", in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper TUPPC004, pp. 554–557. https://jacow.org/ICALEPCS2013/papers/TUPPC004.pdf