Tracking Dubious Data: Protecting Scientific Workflows from Invalidated Experiments

Jim Pruyne,* Justin M. Wozniak,* Ian Foster*
* Data Science and Learning, Argonne National Laboratory, Lemont, IL, USA

***Abstract*—Provenance systems automate record keeping so that humans and/or machines can determine how a given result was obtained. In so doing, they enable a variety of reproducibility and reconstruction capabilities, while tracking the impact of older artifacts on newer ones. Large-scale scientific experiments are increasingly relying on workflows and other automation techniques to keep up with data-rates and perform on-line computation, notably training of machine learning models, and to provide rapid feedback to experimentalists. However, these workflows pose the challenges of: 1) adapting to errors in the experimental process both at the experiment site as well as in computation and 2) complex data provenance patterns that can result from the use machine learning and other methods that can arise from a feedback pattern in which initial experimental results drive the creation of new experimental parameters. The Braid Provenance Engine (Braid-DB) addresses this domain by integrating with workflow systems used in large-scale science and providing the additional capability to drive additional workflows or other automation in response to errors or other causes for elements of the workflow to be considered invalid. In this paper, we describe how Braid-DB responds to data marked as invalid, a common case in experimental science, and demonstrate its ability to retain artifacts unaffected by the invalid data.**

I. Introduction

Large experimental facilities increasingly employ sophisticated computational workflows to perform automated analysis. These workflows may use computational simulations to associate observed data with internal model parameters, train Machine learning (ML) models on experimental and simulation data after various levels of manipulation and processing, and use trained models to control subsequent experiments: for example, by determining when sufficient data has been collected, and ultimately what experiments to run next. As a consequence, machines increasingly support humans as decision makers, and in some cases, short-cut the decision loop so as to accelerate the discovery process.

Such environments pose the question of how to support reproducibility and validation when the *physical* experimental processes used to generate data are affected by ML-based workflows that themselves depend on other data. Decisions such as, how, when, and where to analyze and retain experimental data, and when and how to alter and refine experiment configurations, can have profound impacts on downstream data products. It is thus essential to automate key parts of the record-keeping process so that humans and/or machines can determine how particular results were obtained—and, when problems occur, examine possible causes and take appropriate actions. Reproducing such results requires the ability to record a detailed trace of the ML-driven decisions made and the origins of the processes and models used in making those decisions.

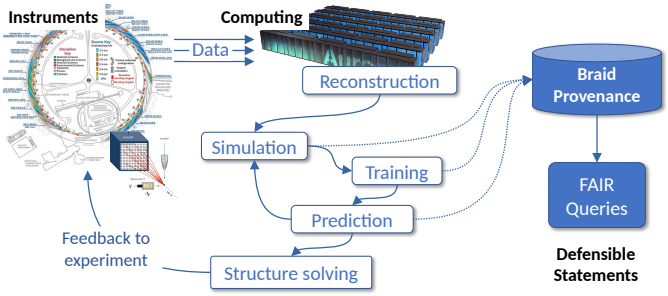

Fig. 1. Braid workflows and their connection to Braid-DB. The Braid provenance engine captures events from Braid tools or as a result of API calls, and supports FAIR principles by recording the provenance of data products in the context of experiments, simulations, and learning.

Physical experiments, as well as computational experiments, are prone to erroneous design, misconfiguration, and faulty execution. Researchers may identify such *dubious experiments* immediately, or only later in an analysis or verification process. In either case, it is essential that the resulting *dubious data* be isolated from the data used to support the defensible scientific statements that result from a study.

As ML-based automation methods become more widely used in research, this task of isolating dubious results becomes more challenging, as a ML model trained on dubious data can influence subsequent steps in complex ways. As depicted in Figure 1, data from an experimental facility may proceed through various workflows before feeding back to the experiment itself, influencing subsequent downstream data. Such data flows may be structured using loops or recursive patterns, for example when raw datasets are augmented by ML model inferences that depend on training data which may, in turn, be based on other data flows. Subtle changes in experimental protocol and policy may have additional effects on downstream data. The impact of dubious data on subsequent discovery steps is difficult to isolate.

To address such challenges, we have designed the Braid Provenance Engine (Braid-DB) to maintain the provenance structure of coarse-grained data products associated with scientific workflows. In this paper, we expand on the scope of our prior discussion [1] by considering the ability to insure that results are derived from valid inputs, and thus a system that can defend scientific statements and FAIR (§II) queries. In the presence of data that are later marked invalid, due to newly realized experimental or computational problems, all affected

data products can be identified, and automated actions can be taken to restore the validity of the overall structure.

Contribution: By developing a conceptual model and implementation of an ML-aware provenance structure with the capability for invalidation, we intend to support the reproducibility and reusability of workflows and their data products. Provenance data of this sort makes datasets more *reusable* by offering a validation structure for the data products, and makes it easy to demonstrate that the data has not been invalidated by implication. It enhances workflow *reproducibility* by capturing, for example, the ML decisions made, so that subsets of the overall experiment can be reproduced, for example, just the physical observations.

Organization: The remainder of this paper is as follows. In §II, we provide background on provenance systems and other related efforts. In §III, we present real-world experimental science applications that can benefit from advanced provenance techniques and describe a representative workflow based on these used in our development efforts.. In §IV, we describe our approach in more detail, and describe our invalidation mechanism. In §V, we describe the Braid-DB software platform and its use. In §VI, we describe how invalidation is integrated into a functional workflow. In §VII, we summarize the paper and provide concluding comments.

II. Background

Provenance systems have been widely studied [2], and there is a significant body of prior work in provenance for computing workflows. These provenance models are commonly designed to capture key aspects of a dataset or artifact, including its derivation history from other artifacts, quality, and ability to be replicated or reproduced [2]. Additional developments are needed to address the needs of workflows coupled with experimental science infrastructure with a goal to capture not only data and associated metadata but also derived models in ways that are Findable, Accessible, Interoperable, and Reusable (FAIR) [3], particularly FAIR principle R1.2 which states that "(Meta)data are associated with detailed provenance."

The Open Provenance Model (OPM) [4] defines provenance concepts without regard to the underlying technologies or systems used to represent them. These concepts are connected with various kinds of edges that represent various types of dependencies. For a particular workflow, the result is an *OPM graph* that captures the causal dependencies among all data products. Souza et al. [5] considered provenance management for ML workloads in sciences. They defined human actors in the ML work cycle, and integrated model training data and other ML-specific concepts, such as ML hyperparameters, into the provenance model. This study, however, did not consider the evolving versions of an ML model that may be produced during an iterative study, in which different ML predictions are produced over time by varying versions of a model. This is a key contribution of the Braid-DB model and is critical for the integration of provenance in experiment-oriented computing.

Polyzotis et al. [6] similarly identified human factors as critical in the ML pipeline. This study considered aspects of raw data manipulation needed for ML training, including cleaning and enriching datasets for training. Metadata tracking schemes for ML workloads have been proposed previously. In ModelDB [7], a ML metadata and training data architecture is easily accessible from within ML-oriented programming environments, including a graphical front-end. Notably, ModelDB, like our Braid-DB, uses a branching history model to track changes over time. An Amazon prototype [8] develops a formal database schema for tracking models and their training data, based on previous but more narrowly defined models [9]. This schema allows for the creation of graphs of data transforms in the ML workflow. Neither system, however, explicitly tracks versions of models in model-model interactions or after iterations of model-experiment iterations.

III. Case studies in Experimental Science

In this section, we summarize a set of large scale scientific applications [10] which link scientific instruments at the Argonne Advanced Photon Source (APS) [11] and Stanford Synchrotron Radiation Lightsource (SSRL) [12] beamlines and high-performance computing at the Argonne Leadership Computing Facility (ALCF) [13].

X-Ray Photon Correlation Spectroscopy (XPCS): This experimental technique is used at synchrotron light sources to study materials dynamics at the mesoscale/nanoscale by identifying correlations in time series of area detector images [14], [15]. The flow comprises 11 steps: 1) transfer of experimental data from APS to ALCF; 2) metadata extraction from the experiment data; 3) transfer of these metadata to persistent storage; 4) index the metadata into a catalog for future query and discovery of data; 5) pre-allocation of computing resources to reduce the risk of unexpected queuing delays for high-performance compute resource; 6) application-specific correlation analysis function which is matrix-heavy is best run on a GPU; 7) plotting of data correlations and reduced size images for presentation in the discovery portal; 8) metadata extraction from computation results; 9) bundle all information generated in the ALCF environment in the previous steps; 10) transfer of the the bundle to long term storage; and 11) augment the indexed metadata in the catalog with generated information

Ptychography: This coherent diffraction imaging technique can image samples with sub-20 nm resolutions [16]. A sample is scanned with overlapping beam positions while corresponding far-field *diffraction patterns*, 2D small-angle scattering patterns containing frequency information about the object, are collected with a pixelated photon counting detector. The flow performs 2D inversion and phase retrieval on diffraction patterns with the following steps: 1) transfer data from APS to ALCF; 2) compute the diffraction for each pattern to obtain a full image; and 3) transfer intermediate results back to APS.

High Energy Diffraction Microscopy (HEDM): This non-destructive technique combines imaging and crystallography algorithms to characterize polycrystalline material microstructure in three dimensions (3D) and under various in-situ thermo-mechanical conditions [17], [18]. The flow comprises these steps: 1) transfer data from APS to ALCF; 2) process each raw image; 3) extract metadata from files and generate

```
# Define Flow steps
transfer_step = Transfer(source="$.data_origin_location",
                         destination="$.compute_data_location")
train_step = TrainModel(input="$.compute_data_location",
                        output="$.model_location")
extract_step = ExtractMetadata(input="$.compute_data_location",
                               result_path="$.ExtractedMetadata")
index_step = IndexMetadata(index_id="$.index_id",
                           content="$.ExtractedMetadata")
return_transfer_step = Transfer(source="$.model_location",
                                destination="$.result_repository")

class BraidFlow(GladierBaseClient):
  # Execute the Flow steps
  steps = [transfer_step, train_step, extract_step,
           index_step, return_transfer_step]

# Define Flow input parameters
flow_input = {"data_origin_location": ...,
  "compute_data_location": ..., "model_location": ...,
  "index_id": ..., "result_repository": ...}

# Launch the flow
run_id = BraidFlow().run(flow_input)
```

Fig. 2. *Gladier* implementation of the example use case

visualizations; 4) process each set of processed images (from step 2) to refine structure; 5) gather metadata; 6) transfer metadata to long-term storage; 7) index raw data, metadata, and visualizations; and 8) transfer the results back to the APS.

A. A Representative Scenario

Based on our experience with these science workflows and the Globus Flows workflow service, we have defined a representative scenario for our experiments and for further discussion here. We define the Flow as follows using the Gladier [19], [20] Python library for defining and executing Globus Flows though for more complex flows, particularly those that include branching, the service's native (and more verbose) JSON syntax may be preferred.

Similar to many of the Flows described above, this Flow: 1) uses Globus Transfer to copy data from a source to a computational environment; 2) uses funcX to perform a computation (in this case we model as a ML model training step); 3) uses funcX to analyze the data and extract metadata information, which it stores in the run-time state of the flow (at the location indicated using the `result_path` argument); 4) indexes the metadata from the Flow's state to a Globus Search index (as defined by `index_id`); and 5) uses Globus Transfer to copy the output of the computation to a final repository. In Figure 2 these steps are shown as the creation of objects from the Gladier library representing the various steps which are then enumerated as `steps` on a class (`BraidFlow`) representing the entire workflow. The listing also shows Gladier use to instantiate and then run the Flow.

IV. Approach

We now describe how the Braid Provenance Engine maintains a provenance structure for artifacts in an environment, notably steps within a Flow and their inputs and outputs, and how elements of this structure may be marked as invalid and how the system acts upon such invalidations.

A. System overview

Braid-DB, like traditional provenance systems (§II), captures computation, such as ML model training or workflow steps and the inputs, outputs and other artifacts associated with the process. Importantly, it integrates provenance records with model version history so as to record how a possibly complex ensemble of variably-accurate models were trained and updated over time. As training data may come from experiment, simulation, or other models (e.g., a higher-accuracy, higher-cost model, or an ensemble of lower-cost models) Braid-DB includes structures that allow a user to ask how a model inference result was obtained, in the form of a defensible *statement*. The provenance structure is designed to be portable so that provenance histories can be merged with other records from collaborating teams, as when a user borrows a model from another team.

BraidRecord: A base-class for all Braid-DB provenance records. Each such entity has a unique ID, a (possibly not unique) string name, a list of URIs representing, and possibly pointing to, the actual data, a list of dependent records (i.e. the provenance relationship), and a dictionary of user-specified, string-keyed metadata tags. Operations are provided to lookup a record based on any of the associated values (ID, name, uri, tags) and for associating another record as a dependency of another record. When defining dependencies, the system insures that the a Directed Acyclic Graph (DAG) semantic is maintained and no cycles are introduced.

BraidFact: A simpler object consisting of static data: for example, pre-existing trusted data or software, etc. BraidFacts may have a provenance outside the Braid-DB system.

BraidData: The Braid-DB representation of traditional provenance-tracked data, with traditional conceptions of its derivation history from other BraidData and/or BraidFacts, such as via simulation.

BraidModel: An ML model tracked by Braid-DB. A BraidModel has the additional capability **update()**, which represents model exposure to other BraidRecords, possibly including other models. This includes the possibility of dependency cycles that capture complex interactions among models and data as experiment workflows progress. These exposures are timestamped, so that queries can determine what a model knew and when the model knew it.

B. Invalidation

Provenance systems can, and often are, used for tracking down errors and their consequences. Braid-DB makes marking records invalid an explicit operation which can, in turn: 1) cause dependent records to be further marked invalid; and 2) automatically cause compensating or recovery workflows or other automation processes to be initiated.

Any BraidRecord may be invalidated with an operation which contains a *cause* description which will be associated with the invalidation allowing further understanding of record validity over time. By default, invalidations *cascade*, meaning that any records which have provenance from a preceding record will also be marked invalid by the operation. In

the cascade case, the invalidation cause description will be applied to all records, but additional references are maintained indicating which preceding record's invalidation resulted in the invalidation of a child record.

In anticipation of invalidation, any record may have associated with it an *Invalidation Action* which will be invoked should the record be invalidated. In the present implementation, the invalidation action takes a form similar to the steps in our workflow system (§V-A) and thus can perform any operations a workflow may. for example, an affected record may be simply annotated instead of being deleted, if the effect of the invalidated data is likely to be negligible. Values from the BraidRecord as well as from the invalidation operation may be used to parameterize the action when it is invoked.

V. Execution Platform

In this section, we describe the services and database structure to support the the production workflows described above and our provenance model, and provide a complete example.

A. Cloud-hosted Service

Globus Flows [21] is a cloud-hosted and professionally operated workflow service providing the reliability, security and scale necessary for production flows supporting large-scale scientific processes. A flow specification in this service consists of a sequence of *Action* steps defining the network location (URL) where an *Action Provider (AP)* may be invoked to perform the operation associated with the step. Each Action Provider implements a Globus Flows-defined API for starting and monitoring the Action. The Flows service implements Action Providers for the following operations: 1) data transfer and management (lookup/delete/access control) via the Globus Transfer service [22]; 2) execution of user-defined functions, including for high-performance computing usage via funcX [23]; 3) indexing and lookup of metadata; 4) E-mail based notification to users; 5) generation of Digital Object Identifiers (DOIs); and 6) web-based forms for users to direct a flow while it is running Each Flow deployed to the service also supports the AP interface allowing Flows to invoke other Flows as Actions, and the open nature of the API allows anyone to host APs for use in their own Flows or for use by others without modification to the Globus Flows service. The interface allows Actions (and the Flows invoking them) to run for as long as weeks or even months allowing very long term operations including human-in-the-loop. Further, end-to-end authentication and authorization is performed such that a user invoking a Flow provides credentials indicating their identity and consent to run the Flow and those credentials are propagated to each AP for further validation.

Flow *runs* are stateful, and each step of the Flow may reference elements of the state (e.g. to provide parameters for a specific Action invocation) and Actions' return values are incorporated into the state of the run allowing outputs from one step to be used as inputs to subsequent steps. Branching steps may check conditions on the current state of the run and select next step based on them. In a typical use case, the developer will simply insert steps into the flow that operate on the Braid-DB, as described in the following.

```
#  Define automated actions upon invalidation
flow_invalidation = InvalidationAction("send_notification", ...)
index_invalidation = InvalidationAction("search_update", ...)

#  Define Flow steps
initial_step = FlowProvenance(invalidation=flow_invalidation)
transfer_step = TransferProvenance(source=...,dest=...,
                                   prev_step=initial_step)
train_step = ComputeProvenance(TrainModel(...),
             prev_step=transfer_step,  input_from_prev=True)
extract_step = ComputeProvenance(ExtractMetadata(...),
         prev_step=train_step, input_from_step=transfer_step)
index_step = IndexMetadataProvenance(
    invalidation=index_invalidation,
    index_id=..., content=..., prev_step=extract_step,
  )

class ProvenanceBraidFlow(GladierBaseClient):
  #  Execute the flow steps as Gladier tools
  gladier_tools = [initial_step, transfer_step,
                   train_step, extract_step, index_step]
```

Fig. 3. Example augmented flow with invalidation

B. Braid-DB implementation

The core of the Braid-DB implementation is a Python library which makes use of an Object Relational Model (ORM) approach to map an object-oriented programming interface for the various Braid-DB abstractions to a relational database. The Braid-DB library, in turn, uses Globus' Python libraries to make authenticated requests to services, notably the Globus Flow service, to carry out invalidation actions when they are triggered. The core library is used in various modes in our experiments. Most notably, funcX-compatible wrappers are provided for operations on Braid records, invalidation actions and invalidation operations in a manner which makes them invokable as steps in Globus Flows. Command line tools are also provided for these operations as well as for generating visual representations of the state of the Braid-DB. The core software libraries can also be used directly for use cases that do not involve workflows or Globus services.

VI. Integration

We developed *augmented* wrappers around typical Flows operations that populate Braid-DB with information on workflow steps and the inputs consumed and outputs generated by these steps. An example augmented flow is shown in Figure 3. For the Flow's run itself (`FlowProvenance`) and for each of the types of operation defined in the original Flow, Transfer (`FlowProvenance`), computation via funcX (`ComputeProvenance`), and metadata indexing (`IndexMetadataProvenance`), wrappers around the existing Gladier classes have been created. These wrappers are used as one-to-one replacements of existing Flow definition elements making transition relatively straightforward. The wrappers take additional provenance information specifying the provenance dependency among the flow steps (the `prev_step` parameter), the input and output artifact dependencies specifying either that a step depends on outputs from the previous step (parameter `input_from_prev`) or a different step (`input_from_step` which may be a single value or a list of values).

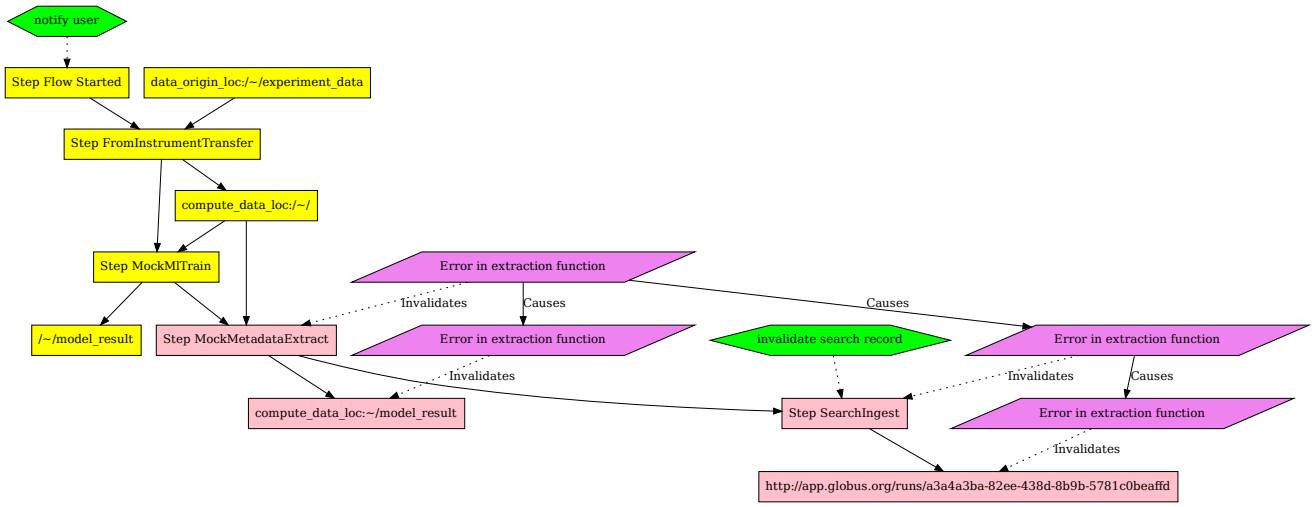

Fig. 4. A graph of provenance information captured from running the representative flow. Rectangular nodes represent steps of the Flow and data or other artifacts. Hexagonal nodes indicate Invalidation Actions and dotted arcs show the record those actions are associated with. Pink parallelograms represent invalidation records with arrows indicating which record they invalidate as well as arcs to other invalidations generated due to invalidation cascading.

Braid-DB invalidation actions are created as objects whose initialization parameters match the required values in the Braid-DB. Instances of these classes may be associated with instances of any of the provenance classes by adding an `invalidation` parameter to their creation. This will then add steps to the generated Flow for creating the invalidation action in the Braid-DB as well as associating that action with the record representing that step of the Flow.

Running the flow in Figure 3 will generate a provenance structure as shown in Figure 4. We have also invalidated record `ExtractMetadata` which is shown with the re-coloring of that and dependent records as well as invalidation records referencing each of the invalidated items. The `IndexMetadata` step has an action associated with it to update the entry in the metadata index. This action will have run as a result of invalidation so that any queries to the metadata index will reflect that this entry is no longer considered to be reliable.

VII. Conclusion and Future Work

We have discussed the importance of workflow-based automation to large-scale instrument-based science, and further how new provenance methods are needed to track next-generation experiments and to adapt to changes or errors in data or workflow processing. These provenance structures are needed to capture not just the history and progress of workloads driven by data capture on these instruments, but also the feedback of results, including those from ML modules into further iterations of an on-going experiment.

The Braid-DB provenance store is tailored for tracking experiments built on workflows and enabling adaption to and compensation for errors in data, computation or other elements that compose the end-to-end scientific process. Braid-DB combines traditional provenance concepts with new concepts relevant to workflow and other automation systems such that provenance information can be used to both propagate changes in state of workflow and data components while requiring only localized knowledge of necessary corrective actions, such as re-training an ML model when errors or other conditions upon which previous results are based are detected. We showed how it can be applied in a workflow representative of those we've encountered in working with our intended user-base with relatively minimal changes to an existing workflow..

Our next step is to move beyond our representative workflow and bring the Braid-DB into production experiment workflows. We will work with the scientists to determine how they get greatest benefit from seeing provenance from their experiments and how that enables them to repeat and evolve their experiments. New tools allowing easy or automatic marking of invalid provenance elements and their corresponding correctie steps will be developed. We will also be pursuing methods to more transparently apply provenance capture to workflows based on Globus Flows such that changes to existing Flows can be minimized or eliminated as well as providing a cloud-based, secure, multi-tenant deployment of the Braid-DB implementing the AP interface so it is available to all of our users without any self-hosting.

References

[1] J. M. Wozniak, Z. Liu, R. Vescovi, R. Chard, B. Nicolae, and I. Foster, "Braid-DB: Toward AI-driven science with machine learning provenance," in *Proc. Smoky Mountains Conference*, 2021.

[2] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *ACM Sigmod Record*, vol. 34, no. 3, pp. 31–36, 2005.

[3] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne *et al.*, "The FAIR guiding principles for scientific data management and stewardship," *Scientific Data*, vol. 3, no. 1, pp. 1–9, 2016.

[4] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephank, and J. V. Busschef, "The Open Provenance Model core specification (v1.1)," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743–756, 2011.

[5] R. Souza, P. Valduriez, M. Mattoso, R. Cerqueira, M. Netto, L. Azevedo, V. Lourenço, E. F. de S. Soares, R. Melo, R. Brandão, D. Salles Civitarese, E. Vital Brazil, and M. Ferreira Moreno, "Provenance data in the machine learning lifecycle in computational science and engineering," in *Workshop on Workflows in Support of Large-Scale Science at SC*, 11 2019, pp. 1–10.

[6] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data management challenges in production machine learning," in *2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1723–1726. [Online]. Available: https://doi.org/10.1145/3035918.3054782

[7] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia, "ModelDB: A system for machine learning model management," in *Proc. Workshop on Human-In-the-Loop Data Analytics*, ser. HILDA '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2939502.2939516

[8] S. Schelter, J.-H. Böse, J. Kirschnick, T. Klein, and S. Seufert, "Automatically tracking metadata and provenance of machine learning experiments," in *Machine Learning Systems Workshop at NIPS*, 2017.

[9] Machine Learning Schema Community Group, "W3C machine learning schema," 2017.

[10] R. Vescovi, R. Chard, N. Saint, B. Blaiszik, J. Pruyne, T. Bicer, A. Lavens, Z. Liu, M. E. Papka, S. Narayanan, N. Schwarz, K. Chard, and I. Foster, "Linking scientific instruments and HPC: Patterns, technologies, experiences," 2022. [Online]. Available: https://arxiv.org/abs/2204.05128

[11] "Advanced Photon Source," https://www.aps.anl.gov. Accessed April 8, 2022.

[12] "Stanford Synchrotron Radiation Lightsource," https://www-ssrl.slac.stanford.edu. Visited April 8, 2022.

[13] K. Riley, M. E. Papka, J. Collins, N. Heinonen, B. Cerny, H. Kim, and L. Wolf, "2019 Argonne Leadership Computing Facility science report," Argonne National Laboratory, Lemont, IL, USA, Tech. Rep., 2019.

[14] O. G. Shpyrko, "X-ray photon correlation spectroscopy," *Journal of Synchrotron Radiation*, vol. 21, no. 5, pp. 1057–1064, 2014.

[15] F. Lehmkühler, W. Roseker, and G. Grübel, "From femtoseconds to hours–measuring dynamics over 18 orders of magnitude with coherent x-rays," *Applied Sciences*, vol. 11, no. 13, p. 6179, 2021.

[16] A. M. Maiden, M. J. Humphry, F. Zhang, and J. M. Rodenburg, "Superresolution imaging via ptychography," *JOSA A*, vol. 28, no. 4, pp. 604–612, 2011.

[17] J. V. Bernier, N. R. Barton, U. Lienert, and M. P. Miller, "Far-field high-energy diffraction microscopy: A tool for intergranular orientation and strain analysis," *The Journal of Strain Analysis for Engineering Design*, vol. 46, no. 7, pp. 527–547, 2011.

[18] R. Pokharel, "Overview of high-energy x-ray diffraction microscopy (HEDM) for mesoscale material characterization in three-dimensions," in *Materials Discovery and Design*. Springer International Publishing, 2018, pp. 167–201. [Online]. Available: https://doi.org/10.1007/978-3-319-99465-9_7

[19] Gladier Team, "Gladier software," 2021, https://github.com/globus-gladier.

[20] ——, "Gladier client templates," 2021, https://github.com/globus-gladier/gladier-client-template.

[21] R. Chard, J. Pruyne, R. Richter, U. Mandujano, K. McKee, S. Thompson, J. Bryan, B. Raumann, R. Ananthakrishnan, K. Chard, and I. Foster, "Research process automation across the space-time continuum," *arXiv preprint*, 2022.

[22] K. Chard, S. Tuecke, and I. Foster, "Efficient and secure transfer, synchronization, and sharing of big data," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 46–55, 2014.

[23] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard, "FuncX: A federated function serving fabric for science," in *29th International Symposium on High-Performance Parallel and Distributed Computing*. New York, NY, USA: Association for Computing Machinery, 2020, p. 65–76. [Online]. Available: https://doi.org/10.1145/3369583.3392683

VIII. Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357.

The following text will be removed in the final submission: