2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)

Provenance Data in the Machine Learning Lifecycle in Computational Science and Engineering

Renan Souza[§,o], Leonardo Azevedo[§], Vítor Lourenço[§], Elton Soares[§],
Raphael Thiago[§], Rafael Brandão[§], Daniel Civitarese[§], Emilio Vital Brazil[§],
Marcio Moreno[§], Patrick Valduriez[#], Marta Mattoso[o], Renato Cerqueira[§], Marco A. S. Netto[§]
[§]IBM Research, [o]Federal Univ. of Rio de Janeiro, [#]Inria & LIRMM, U. Montpellier

***Abstract*—Machine Learning (ML) has become essential in several industries. In Computational Science and Engineering (CSE), the complexity of the ML lifecycle comes from the large variety of data, scientists' expertise, tools, and workflows. If data are not tracked properly during the lifecycle, it becomes unfeasible to recreate a ML model from scratch or to explain to stakeholders how it was created. The main limitation of provenance tracking solutions is that they cannot cope with provenance capture and integration of domain and ML data processed in the multiple workflows in the lifecycle, while keeping the provenance capture overhead low. To handle this problem, in this paper we contribute with a detailed characterization of provenance data in the ML lifecycle in CSE; a new provenance data representation, called PROV-ML, built on top of W3C PROV and ML Schema; and extensions to a system that tracks provenance from multiple workflows to address the characteristics of ML and CSE, and to allow for provenance queries with a standard vocabulary. We show a practical use in a real case in the O&G industry, along with its evaluation using 48 GPUs in parallel.**

***Index Terms*—Machine Learning Lifecycle, Workflow Provenance, Computational Science and Engineering**

I. Introduction

Machine Learning (ML) has been fundamentally changing Computational Science and Engineering (CSE) in various ways [1]. Techniques, such as statistical relational learning and deep learning, have been used to extract knowledge from data, with application domains ranging from Computational Physics to Agriculture and Oil and Gas (O&G) [2–5]. Obtaining reliable ML models involves a complex ML lifecycle [6], which is critical in large-scale CSE projects [2, 7]. The *ML lifecycle in CSE* depends on transforming raw data into trained models, which requires multiple, distributed workflows that use a wide variety of algorithms, data, data processing tools and data stores; demands execution in machines ranging from standalone servers to cloud or HPC clusters; and is carried out by multidisciplinary teams, including domain scientists, computational scientists and engineers, and ML specialists. Given the heterogeneous nature of the lifecycle, it is difficult to track, in an integrated way, the data transformations that occur throughout the lifecycle while keeping the execution overhead low, which is a major concern among CSE users. In practice, tracking the data in the data transformations is often done manually, which is time consuming and error prone. This is problematic for several reasons, ranging from scientific (*e.g.*, jeopardizes reproducibility) to business (*e.g.*, users may be less likely to apply a trained model, even with best performance, if they do not understand the transformations in the lifecycle).

Data lineage (*i.e.*, data provenance) helps reproducing, tracing, assessing, and understanding data and their transformation processes [8]. Solutions for provenance data tracking for ML have been proposed [9–11], but with focus on learning phases only, thus limiting an integrated view of domain-specific data, processed in the early phases of the lifecycle, with ML data. Besides, users need to migrate their workflows to a different software ecosystem or change the way they develop, which may compromise adoption in CSE. Another approach is to add provenance tracking to workflows, reducing the need to change the development practice [12–16]. Nonetheless, solutions following this approach do not support the lifecycle, which requires three main capabilities: provenance tracking in multiple workflows that use heterogeneous data and stores; a provenance data representation with ML-specific vocabulary; and providing for integrated data analysis through provenance while keeping the capture overhead low. Another solution following this approach is ProvLake [17], which has low capture overhead in multiple workflows that use heterogeneous data stores. However, similarly to the other solutions, its provenance data representation is based on W3C PROV only, thus lacking ML-specific vocabulary, limiting its support for the ML lifecycle in CSE. Allowing for such provenance analysis that integrates both ML data and domain-specific data while keeping capture overhead low in HPC workflows is hard.

In this paper, we propose an end-to-end solution for tracking data transformations that occur in the ML lifecycle in CSE, from the curation of raw data until the generation of trained models, by providing efficient provenance capture and data analysis through provenance queries. Our approach is to model the lifecycle as multiple workflows interconnected with data and to track provenance as the workflows execute. By adding provenance capture calls in the workflows, users can perform ML monitoring (*e.g.*, the evolution of model performance as the training iterates) and more comprehensive provenance analyses that join domain-specific data with ML data generated in the lifecycle. The main contributions of the paper are:

(i) a characterization of provenance data in the ML lifecycle in CSE (Sec. II);

(ii) PROV-ML: a new data representation, which combines W3C PROV [18] with W3C ML Schema [19], for prove-

978-1-7281-5997-3/19/$31.00 ©2019 IEEE
DOI 10.1109/WORKS49585.2019.00006

nance of the ML lifecycle in CSE (Sec. III);

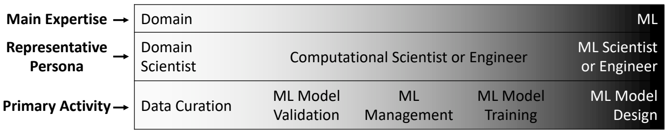
(iii) extensions of the ProvLake [17] to support provenance tracking and analysis following the PROV-ML (Sec. IV). Experiments show its practical use in a real O&G case in a testbed of 48 GPUs (Sec. V).

II. Workflow Provenance for ML in CSE

This work focuses on provenance in workflows formed by chained data transformations composing the ML lifecycle in CSE, aiming at supporting the data analysis in a large-scale CSE project. Before we characterize the data analysis through provenance (Sec. II-C), we first characterize the lifecycle's personas (Sec. II-A) to provide for analysis addressing the users' needs, then we describe the lifecycle (Sec. II-B).

A. Personas in the ML Lifecycle in CSE

Large-scale CSE projects are often multidisciplinary, with collaborating users with different skills on the domain data, *e.g.*, mathematics, physics, statistics, computational methods, and ML. These users perform distinct types of analysis and have different provenance requirements. In order to position the personas and their primary activities in the ML lifecycle in CSE, we adapt background work on traditional scientific workflows [20] and ML [6]. Fig. 1 illustrates how expertise, representative personas, and primary activities fall under an expertise spectrum ranging from scientific-domain only (fully white on the left) to ML only (fully black on the right).



Main Expertise →	Domain				ML
Representative Persona →	Domain Scientist	Computational Scientist or Engineer			ML Scientist or Engineer
Primary Activity →	Data Curation	ML Model Validation	ML Management	ML Model Training	ML Model Design

Fig. 1: Spectrum of expertise and personas in the lifecycle.

Domain scientists. They have in-depth knowledge of the domain data and use specialized tools to interpret, visualize, and clean the scientific data [16], thus playing an important role to curate the raw scientific data, specify domain matters, and validate results. Examples are geoscientists, agronomists, and experimental physicists. They can validate ML models qualitatively, *i.e.*, they can check if results are reasonable given the characteristics of the domain. Oftentimes, such validations contradict the numeric results obtained by ML algorithms. They are also paramount in hypothesis definition and in verifying simplifications made about domain's problems. They contribute by collecting domain-specific annotations from technical reports and articles, and link such annotations to the raw scientific data, augmenting the possibilities for enhanced analyses, contributing to the training. They are critical to providing labeled scientific data to supervised learning algorithms.

Computational scientists and engineers. They have high computational skills, often with abilities to develop parallel scripts and execute them in HPC clusters. Examples are computational physicists, engineers. They are highly knowledgeable in the domain, although not as in-depth as the domain scientists. They are familiar with traditional numerical simulations that require HPC, which need complex scientific data analyses to guide the fine-tuning of parameters [1, 16, 21]. In the lifecycle, they are often the ML model trainers, who tune parameters, a very usual task when training ML models. They use their knowledge on the domain to make decisions, *e.g.*, to filter relevant parts of the training datasets that are guaranteed to respect the physical constraints of the problem. Some users with more in-depth knowledge of ML techniques, *i.e.*, those who are more towards the black portion of the spectrum in Fig. 1, can design new ML models. They can, for instance, design Physically-informed Neural Networks (PINNs) [3], which embed domain-specific physical constraints in the ML models. These users can be responsible for validating the ML model and, more experienced users with considerable ML and domain knowledge, help in the overall analyses of the produced models, their quality, how they were used, etc.

ML scientists and engineers. They have in-depth knowledge of statistics, ML algorithms, and software engineering. They design new ML models and develop scripts typically using ML libraries like TensorFlow, PyTorch, and Scikit Learn. They are familiar with software engineering techniques (*e.g.*, continuous integration, test-driven development, cloud deployment) and can use different kinds of DBMSs to store data. They often train the ML models they design, often in HPC clusters. Moreover, to be able to develop effective models, they also have some domain knowledge.

Provenance specialists. In addition to those three main personas, Provenance specialists play an essential role in a CSE project by managing data provenance in the lifecycle. They design the provenance schema for applications and guide other users to add provenance capture calls to the workflows. Thus, they need knowledge in the scientific domain and ML. They also support other personas to analyze provenance, domain-specific, execution, and ML data.

B. The ML lifecycle in CSE

We can divide the ML lifecycle in CSE into three major phases: *data curation*, *learning data preparation*, and *learning* (Fig. 2 — dashed arrows are data flows and solid arrows are interactions between phases). Our view of the lifecycle is inspired by Polyzotis *et al.*' survey [6]. Although they proposed a generic view, which can be applied to CSE, there remains the need for a focused view on the problems inherent to CSE. We grouped the inner phases into major phases, organizing the activities according to the scientific data manipulated and the personas involved in the major phases.

Data curation. It is the most complex phase of the lifecycle, especially because of the nature of the scientific data. To achieve automated knowledge extraction from scientific data promoted by ML, much manual and highly specialized work is performed by the users (mainly domain scientists). There is a huge gap between raw scientific data and useful data for consumption (*e.g.*, data to serve as input to train ML models). Datasets are typically large, up to terabytes in a single file.

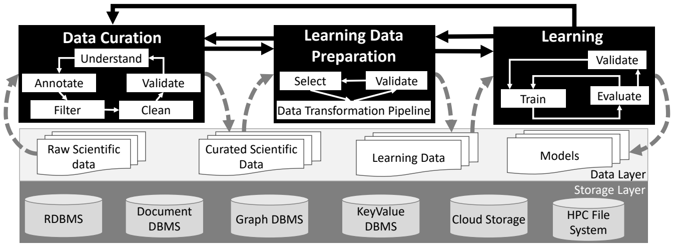


Fig. 2: The ML lifecycle in CSE.

They may contain geospatial-temporal data, stored as huge matrices in well-known scientific formats, like HDF5. Also, some files are stored using domain-specific formats, *e.g.*, SEG-Y format for seismic data, widely used in the O&G industry. Specialized formats in CSE domains may require industry-specific software and domain-specific knowledge to inspect, visualize, and understand the data. In addition, users can use metadata and textual reports to annotate the data with extra domain-specific knowledge, without which would be nearly impossible to make the data useful for ML algorithms.

Considering the heterogeneous nature of the data, "it is unreasonable to assume that data lives in a single source" (*e.g.*, a single file system or DBMS) [6]. For instance, raw files can be stored in file systems or cloud stores, domain-specific annotations can be stored in a Semantic Graph DBMS (*e.g.*, Triple Store) with domain ontologies, and curated data can be stored in a NoSQL DBMS, such as Document DBMSs. Then, computational scientists and engineers write data-intensive workflow scripts to clean, filter, and validate the data. For instance, they check if the geolocalization of the data files is consistent. These scripts transform the raw data into curated data by consuming and generating data from those data stores. Each of these inner phases inside the data curation phase is highly interactive, manual, and may execute independently. In other words, users may run different scripts to execute these phases, several times, in an *ad-hoc* way and any order. Also, they run these scripts in different machines, such as in an HPC cluster or in the cloud, or even on the users' desktop. These phases occur in a cycle, which stops when the users consider the data "curated". These curated data are significantly more organized and easier to analyze and understand. In the context of ML, it is ready to be transformed into training data.

Learning data preparation. Model trainers select relevant parts of the curated data to be used for learning. For instance, if the ML task is to classify geological structures [5], seismic images will need to be correlated with annotations -seismic interpretation-, creating annotated samples. After selecting the data, model designers develop scripts that transform the data into training datasets. Typical transformations include image crop, quantization, scale, among others. In this phase, users frequently use domain-specific libraries to manipulate raw scientific data. Due to data complexity, oftentimes data need to be manually inspected before it can be used as input for the learning phase.

TABLE I: Examples of provenance queries in ML for CSE.

Q1	Given a trained model, what are the geographic coordinates, oil basin and field, and the number of seismic slices of the seismic in the training dataset?
Q2	Given a trained model, what is the tile size, the noise filter threshold, and the ranges of seismic slices that were selected to generate the training set used to adjust this model?
Q3	Given a training set, what are the values for all hyperparameters and the evaluation measure values associated with the trained model with least loss?
Q4	What are the average, min, and max execution times of each batch iteration inside each epoch of the deep neural network training, given a training dataset?
Q5	What is the execution time on average per batch iteration, per epoch, and what are the evaluation metrics of the trained models that used the training dataset generated for a given range of seismic slices?
Q6	Given the training dataset used in Q5, what was the seismic data file used, along with its number of slices, related oil basin, and field?

Learning. In this phase, model trainers select the input training datasets, optionally they choose validation datasets, and choose training parameters (*e.g.*, in deep learning they can choose ranges of epochs and learning rates) that will be optimized in the training process. Trainers can use their domain knowledge to discard input training datasets that will unlikely provide good results. The training process is compute-intensive, typically executed as a job submitted in an HPC cluster. One single training process often generates multiple trained models, among which one is chosen as the "best" depending on evaluation metrics (*e.g.*, MSE, accuracy, or any other user-defined metric). Moreover, as the training process takes a long time, trainers need to monitor it by, *e.g.*, inspecting how the evaluation metrics are evolving while the training process iterates. They can wait until completion or interrupt the training process, change parameters, re-submit the training in an iterative way until satisfied with results.

C. Characterizing Provenance Analysis in ML for CSE

Provenance data in workflows contain a structured record of the data derivation paths within chained data transformations, along with the parameterization of each transformation [15, 21]. Provenance data are usually represented as a directed graph where: vertices are instances of entities (typically data) or activities (typically the data transformations) or agents (typically the users); and, edges are instances of relationships between vertices [18]. Scientists use provenance data for reproducibility and result understanding [8]. This kind of provenance consumption, which often occurs *post mortem*, *i.e.*, after workflow execution, is characterized as offline provenance analysis. A characterization of provenance analysis to leverage ML in support of workflows is surveyed by Deelman *et al.* [7]. We propose here a taxonomy to classify provenance analysis in support of ML, by considering three classes: *data*, *execution timing*, and *training timing*. We provide the characterization based on the data being analyzed, using query examples (listed in Table I) in our use case in O&G.

Use case. The use case addresses seismic surveys, which are indirect measures of the earth subsurface that can be organized into slices (images). These surveys cover hundreds of square kilometers and help to interpret the geology and find possible hydrocarbons accumulations. The seismic data have a very complex workflow and can suffer from many problems, like noise and shadows (regions with low signal). Also, the geological structures vary from point to point in the earth, imposing significant challenges to the ML algorithms. Next, we characterize the data involved in the lifecycle.

Data class includes *domain-specific*, *machine learning*, and *execution*. Provenance data may be augmented with these data, increasing the scope of provenance analysis.

Domain-specific data are the main data processed in the data curation phase (Sec. II-B). Approaches to add domain data into provenance analysis include, *e.g.*, raw data extraction [15] and utilization of semantic domain databases associated to provenance databases [17]. For raw data extraction, quantities of interest are extracted from large raw data files, and for domain databases, domain scientists may provide relevant information and metadata about the raw data and store them in knowledge data graphs (*e.g.*, in Triple Stores).

Machine learning data include training data and generated trained models, which are more related to the learning data preparation (*e.g.*, Q1) and learning (*e.g.*, Q2 and Q3) phases (Fig. 2). These queries exemplify that the parametrization within the data transformations and relevant metadata of the generated data (both training data and trained model) are important for provenance analysis.

Execution data. Besides model performance metrics (*e.g.*, accuracy), users need to assess execution time and resources consumption of their workflows. They need to inspect if a critical block in their workflow (*e.g.*, the one that demands high parallelism) is taking longer than usual or if other parts are consuming more memory than expected. For this, provenance systems can capture system performance metrics and timestamps (*e.g.*, Q4). Metadata, such as data store metadata (*e.g.*, host address), HPC cluster name and nodes in use, can be captured and associated with the provenance of the data transformations for extended analysis.

Hybrid. Users can combine these data. For instance, in Q5, the analysis queries data processed in workflows in the learning data preparation and learning phases, whereas Q6 uses the same data generated in the learning data preparation to analyze the raw files curated in the data curation phase.

Execution timing refers to if the analysis is done *online*, *i.e.*, while at least a workflow is running, or *offline*.

Offline analysis. The typical use of offline provenance analysis is to support reproducibility and historical results understanding, *e.g.*, understand the data curation phase of raw scientific files and relate with the generated trained ML models. The queries Q1–Q6 can be executed offline.

Online analysis. Users can use online provenance analysis to monitor, debug or inspect the data transformations while they are still running (*e.g.*, see the status, see how the intermediate results are evolving as the input parameters vary). The problem of adding low provenance data capture overhead is more challenging for provenance systems that allow for online analysis [17]. Queries Q3–Q5 exemplify queries that can be executed online, *e.g.*, while a training process is running.

Training timing refers to whether the analysis performs *intra-training*—*i.e.*, to inspect one training process, *e.g.*, a training job running on an HPC cluster, or *inter-training*—*i.e.*, analyses comprehending results of several training processes.

Intra-training. In an offline intra-training analysis, users are interested in understanding how well trained models generated in a given training process perform. All queries, Q1–Q6, could be executed either online or offline, but Q3 and Q4 are more likely to be performed as online intra-training analysis.

Inter-training. This analysis refers to comprehensive queries to understand multiple training processes, *e.g.*, how each of them performed, which training datasets were used, how the training processes were parameterized. This is very important in the lifecycle, as it supports activities like Model Validation, Management, Training, and Design. Usually, they are used offline, but may also be performed online. Queries Q1–Q6 fit this class when analyzing multiple trained models generated in different training processes.

Further characterization. Other classes worth mentioning for provenance analysis for ML in CSE are: *data store*—data are distributed onto multiple stores, like file systems, cloud stores (*e.g.*, IBM Cloud Object Storage), Relational or NoSQL DBMSs [17]; *provenance data granularity*—provenance of files (*i.e.*, references to files consumed and generated in a script), functions calls (arguments and outputs), blocks of code, and stack traces [14]; and *provenance analysis direction: forward or backward*—generally, forward queries analyze from raw scientific files or training datasets to trained models (*e.g.*, Q3–Q5), whereas backward queries analyze from trained models to training datasets or raw files (*e.g.*, Q1, Q2, Q6).

III. ML Provenance Data Representation

There are many workflow provenance tracking solutions [12, 13, 15, 17], but they are often based on W3C PROV [18] (and extensions) only. Thus, they are too generic in terms of provenance data representation and analysis, which makes the adoption for ML more difficult. An existing work on ML data representation is the W3C ML Schema (MLS) [22]. Although the MLS has some provenance representation, a MLS-based only representation does not meet the needs either. It does not have a clear distinction between prospective and retrospective provenance data, which compromises query capabilities because prospective provenance provides the abstraction layer to specify provenance analysis over data generated in workflows' execution (*i.e.*, retrospective provenance). Also, MLS does not separate the learning stages (training, validation and test), which would enable finer analysis based on specific stages. Finally, it is not designed to allow for representation of domain-specific data generated at early phases of the lifecycle (*e.g.*, data curation).

To address these problems, this section introduces PROV-ML. To the best of our knowledge, it is the first provenance

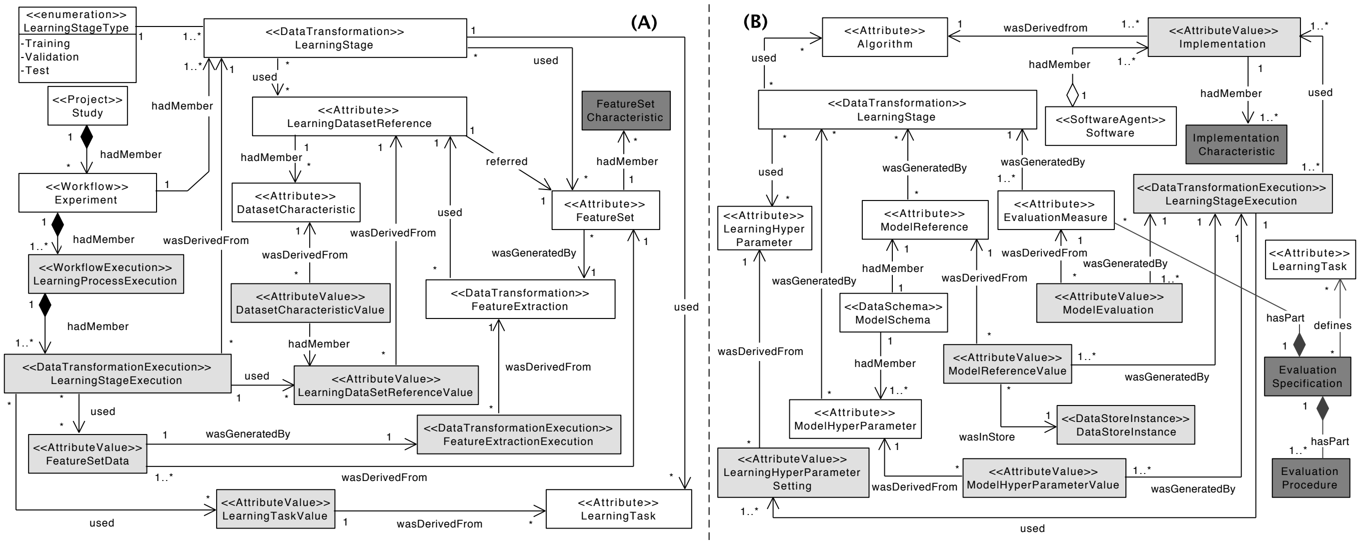

(A)
<<enumeration>> LearningStageType
-Training
-Validation
-Test
<<DataTransformation>> LearningStage
<<Project>> Study
<<Workflow>> Experiment
<<WorkflowExecution>> LearningProcessExecution
<<DataTransformationExecution>> LearningStageExecution
<<Attribute>> LearningDatasetReference
<<Attribute>> DatasetCharacteristic
<<AttributeValue>> DatasetCharacteristicValue
<<AttributeValue>> LearningDataSetReferenceValue
FeatureSet Characteristic
<<Attribute>> FeatureSet
<<DataTransformation>> FeatureExtraction
<<DataTransformationExecution>> FeatureExtractionExecution
<<AttributeValue>> FeatureSetData
<<AttributeValue>> LearningTaskValue
<<Attribute>> LearningTask
hadMember
used
referred
wasDerivedFrom
wasGeneratedBy
(B)
<<Attribute>> Algorithm
<<AttributeValue>> Implementation
<<SoftwareAgent>> Software
Implementation Characteristic
<<DataTransformation>> LearningStage
<<DataTransformationExecution>> LearningStageExecution
<<Attribute>> LearningHyper Parameter
<<Attribute>> ModelReference
<<Attribute>> EvaluationMeasure
<<AttributeValue>> ModelEvaluation
<<DataSchema>> ModelSchema
<<AttributeValue>> ModelReferenceValue
<<DataStoreInstance>> DataStoreInstance
<<Attribute>> ModelHyperParameter
<<AttributeValue>> LearningHyperParameter Setting
<<AttributeValue>> ModelHyperParameterValue
<<Attribute>> LearningTask
Evaluation Specification
Evaluation Procedure
wasDerivedFrom
hadMember
used
wasGeneratedBy
wasInStore
hasPart
defines

Fig. 3: PROV-ML: a W3C PROV- and W3C ML Schema-compliant workflow provenance data representation.

data representation for workflows in the ML lifecycle in CSE. It is compliant with both W3C PROV and MLS, and extends ProvLake's workflow provenance data representation [23], which is an extension of PROV.

PROV-ML provides detailed support for the learning and learning data preparation phases of lifecycle. It inherits the benefits of ProvLake, enabling the integration of provenance of domain-specific data processed by workflows in the curation phase. PROV-ML is depicted in Fig. 3, where Fig. 3(A) shows the relation of the learning phase with the input data and the goal of a ML workflow (*i.e.*, a workflow in the learning phase); and Fig. 3(B) represents the relation of the learning phase with its technique and parameters. Classes in white background represent prospective provenance; light gray, retrospective; and dark gray represents specific concepts inherited, as is, from MLS. PROV-ML classes are described on Table II. Further details on PROV-ML are online [23].

IV. PROVLAKE IN THE ML LIFECYCLE IN CSE

To address provenance tracking and analysis throughout the ML lifecycle in CSE, our approach is to model it as multiple workflows with chained data transformations, where the workflows are interconnected through data. Provenance tracking comprises provenance capture, the creation of the provenance relationships (*e.g.*, associations between the processes and the consumed and generated data), and storage of the provenance data. In our view, provenance tracking systems that can be coupled to workflows [12, 13, 15, 17] provide the flexibility needed in large-scale CSE projects, as opposed to moving workflows' executions and data to be managed by a single orchestration system, like a Workflow Management System. Workflow provenance capture systems usually address scripts as workflows with chained functions, method, or library calls that execute data transformations, while capturing input arguments and output values from these calls. Among these solutions, ProvLake [17] has been applied to capture provenance from multiple distributed workflows that consume and generate data from and to heterogeneous data stores, while keeping provenance capture overhead low. While these workflows execute, provenance data are captured and stored in a single provenance database, available for integrated analysis of the data generated throughout the lifecycle. This section describes ProvLake architecture and deployment in support of the lifecycle.

Architecture. It has five main components (Fig. 4): ProvLake Library (PLLib); ProvTracker; ProvManager; PolyProv-QueryEngine; and Prov DBMS (the DBMS that manages the provenance database).

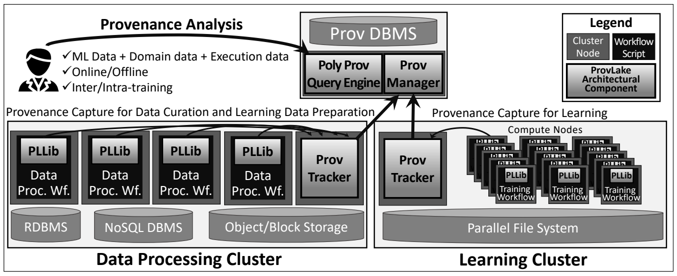


Fig. 4: ProvLake architecture on an exemplary deployment on two clusters: for data preprocessing and for learning.

The workflows are instrumented with PLLib, imported as a library in the scripts, which is responsible for the provenance data capture. In an offline manner and following the methodology described in a previous work [17] to specify the workflows using prospective provenance data standards [8], users add provenance data capture calls using the PLLib. A provenance capture task happens when a data transformation executes, which typically occurs in a function call, in a

TABLE II: PROV-ML data representation classes.

Class	Description
Study	Investigation (*e.g.*, research hypothesis) leading ML workflow definitions.
Experiment	The set of analyses (*e.g.*, research questions), that drives the ML workflow.
LearningProcessExecution	An ML workflow execution. This is equivalent to *mls:Run* and was renamed to explicitly preserve the aspects of retrospective provenance, which are not explicitly handled in MLS.
LearningTask and LearningTaskValue	Defines the goal of a learning process, *i.e.*, the ML task (*e.g.*, *LearningTask*: *Classification*; *LearningTaskValue*: *Seismic Stratigraphic Classification*).
LearningStageType	A stage in the learning process. It is one of: training, testing or validation.
LearningDatasetReference	Defines the dataset to be used by a *LearningStage* and *LearningDatasetReferenceValue* is the dataset reference used in a *LearningStageExectution*.
DatasetCharacteristic and DatasetCharacteristcValue	Defines metadata about the *LearningDatasetReference* (*e.g.*, #instances), and *DatasetCharacteristcValue* relates with a *LearningDatasetReferenceValue* (*e.g.*, #instances =8).
FeatureSet and FeatureSetData	Defines the features *FeatureExtraction* to generate over *LearningDatasetReference* and, *FeatureSetData* is the generated values in the execution.
FeatureSetCharacteristic	Defines the set of metadata that describes the *FeatureSet* (*e.g.*, number of features, features' type).
FeatureExtraction and FeatureExtractionExecution	Defines the features retrieval process.
Software	Defines a collection of ML techniques' implementations (*e.g.*, Scikit-Learn).
Algorithm	ML technique with no associated technology, software or implementation (*e.g.*, k-means clustering technique).
Implementation	Defines the retrospective aspect of an *Algorithm*, *i.e.*, an ML technique's implementation in a software (*e.g.*, Scikit-Learn's k-means implementation).
ImplementationCharacteristic	Defines the implementation's set of metadata, (*e.g.*, version, git hash).
LearningHyperParameter	Defines the prior parameter of an *Algorithm* used by a *LearningStage*.
LearningHyperParameter Setting	Defines the parameter values of an execution (*e.g.*, the k value in a k-means clustering technique, range of epochs in a neural network training).
ModelSchema	The scope of the resulting model.
ModelReference and ModelReferenceValue	The resulting model of a *LearningStage* should generate and the generated value (*e.g.*, the trained model after the training stage).
ModelHyperParameter and ModelHyperParameterValue	Hyperparameters a *LearningStage* generates and the resulting model with their values (*e.g.*, the epoch which the resulting model was generated), respectively.
DataStoreInstance	Resulting model (*i.e.*, *ModelReference-value*) storage.
EvaluationMeasure and ModelEvaluation	A measure a *LearningStage* should evaluate and its associated value generated in execution (*e.g.*, the precision of classifier model).
EvaluationSpecification and EvaluationProcedure	Classes directly inherited from MLS, with their semantics preserved.

program execution, or in an iteration in an iterative workflow. As shown in a simplified pseudocode of a deep learning training (Algorithm 1), a provenance task is delimited within blocks of code in the scripts, illustrated with `prov.in()` and `prov.out()`, each generating a provenance capture event.

PLLib design has two goals: (i) to keep execution overhead low and (ii) to avoid major modifications in the user code while preserving the provenance data analytical capabilities. Because of the workflow specification using prospective provenance data, kept external to the workflow scripts and loaded only once at the beginning (Line 2 in Algorithm 1), the code modification and data to be sent to ProvTracker are reduced. Design principles such as queuing provenance requests, asynchronicity (*i.e.*, the workflow scripts do not wait for the provenance requests to be fully processed—the pipeline from the PLLib to ProvTracker, ProvManager, and Prov DBMS), and reduction of system calls help reducing capture overhead [17]. Provenance capture requests are queued and the maximum queue size is a configurable parameter. Moreover, users choose to store provenance data on disk only, rather than sending to ProvTracker, but in this case, online provenance analysis is not supported. Then, if disk only is not specified, when the scripts execute, provenance data are captured and sent to ProvTracker.

ProvTracker uses prospective provenance data to provide for the tracking by creating the relationships of retrospective provenance data being continuously sent by PLLib, from multiple distributed workflows. ProvTracker gives unique identifiers to every data value captured by the PLLib, so when a data transformation consumes data produced by another, ProvTracker will track such relationship and populate the data graph. When the data values are data references (*e.g.*, references to files or identifiers in a database table or any analogous data reference), it creates an edge between the data value and the data store [17]. Data transformations that are specific and standard in ML workflows, *e.g.*, training, validation, and testing are defined beforehand following PROV-ML (III). ProvTracker also allows users to specify, in the

Algorithm 1: Provenance capture in a training script.

```
   Input: training hyperparameters, input data sets
1  import PLLib as prov
2  prov.init($prospective\_provenance$)
3  ...
4  prov.in($'training'$, $training\_hyperprms$,
     $input\_data\_references$)
5  for $e = 1$ .. $max\_epochs$ do
6      prov.in($'epoch'$, $e$)
7      ...
8      for $batch\_id$ in $data\_batches$ do
9          prov.in($'batch'$, $batch\_id$)
10         ...
11         prov.out($'batch'$, $loss\_value$)
12     prov.out($'epoch'$, $confusion\_matrix$,
         $model\_hyperprms$, $model\_perf$, $model\_ref$)
13 prov.out($'training'$)
14 prov.close()
```

prospective provenance specification, that certain parameters or output values have ML-specific semantics, following PROV-ML, to be stored in the provenance database. Moreover, ProvTracker has work queues to group provenance requests before sending retrospective provenance data to ProvManager. ProvManager is a RESTful service that receives provenance data using PROV-ML vocabulary, and transforms the data into RDF triples (the data model of the DBMS in use by ProvLake in this current implementation) and inserts them in a bulk.

Provenance queries are provided by the PolyProvQueryEngine. The characterization (Section II-C) and typical queries (*e.g.*, Q1–Q6) are used to influence the implementation of parameterized RESTful endpoints using PROV-ML terms. Variations of this endpoint, using terms available in PROV-ML, are used to specify the inputs for the queries. If an endpoint is not implemented for a specific query, users can still write raw queries and submit them to PolyProvQueryEngine directly, which redirects the query to the Prov DBMS.

Execution Strategies on HPC Clusters. ProvLake uses a microservice architecture to achieve high flexibility when specifying how the components are deployed to reduce performance penalties. Fig. 4 shows a deployment of ProvLake onto two clusters, one for I/O-intensive workflows like the data processing ones (*Data Proc. Workflows* in the figure—used for the data curation and learning data preparation phases of the lifecycle) and the other for compute-intensive workflows, like the training workflows (for the learning phase). PLLib is the only component in direct contact with the users' workflows running in the clusters, shielding the workflows from possible slowness from other components. To reduce communication cost between the users' workflow and the PLLib, ProvTracker is deployed inside the cluster. To avoid competition (which increases overhead) with the users' workflows, ProvTracker is started on a separate node in the cluster. The other architectural components are deployed externally to the clusters because they are not in direct contact with the PLLib, thus not increasing the communication cost in the workflow scripts. This avoids using extra computing resources only for provenance tracking and analysis, leaving more resources for the users' workflows; and avoids operational work to install more software, such as a DBMS, inside a compute-intensive cluster.

V. Experimental Validation

In this section, we provide an experimental validation of ProvLake in support of the ML lifecycle in CSE. As execution overhead is a major concern among CSE users, we first present a performance analysis of parallel provenance data capture in Section V-A, then we show a running example of which data are captured during the lifecycle of our case study to answer the exemplary queries Q1–Q6 in Section V-B.

Hardware setup. We use two clusters: a learning cluster, which has 393 Intel and Power8 nodes, each with 24 to 48 CPU cores, 256 to 512 GB RAM, interconnected via InfiniBand, sharing about 3.45 PB in a GPFS, and using in total 946 GPUs (NVIDIA Tesla K40 and K80, each with 2880 and 4992 CUDA cores respectively); and a data processing cluster, which has 12 nodes, each with 128 GB RAM, two Intel CPUs with 40 cores, sharing a GPFS with 24 TB, interconnected via an InfiniBand.

Software setup. ProvManager, PolyProvQueryEngine, and Prov DBMS are deployed on a virtual Kubernetes cluster with two nodes with 4 vCores, 16 GB RAM each, virtualized on top of the data processing cluster. As in Fig. 4, the ProvTracker service is started on a separate node on each of the two clusters. ProvLake's services are implemented using Python and deployed with uWSGI with C++ Cython plugin with multi-process and multi-thread parallelism enabled. ProvManager's queue is set to 50 and ProvTracker threads are set to 120. The workflow scripts of our use case are implemented in Python using multiple libraries, such as to manipulate raw seismic files and for learning (PyTorch V1.1).

A. Performance Analysis

In our use case for training an autonomous identifier of geological structures (*c.f.* Sec II-C), the learning phase generates a large amount of provenance data at a high frequency to stress ProvLake services. In the deep learning model training, there are two provenance capture calls (for the beginning and end) at each batch iteration, in each learning epoch (*c.f.* Algorithm 1). In this test, each training workflow executes about 35 iterations for each learning epoch and up to 300 epochs, generating about 15,000 provenance capture events per workflow run. ProvTracker runs on one node in the learning cluster with 24 CPU cores, whereas the training workflows run in parallel and distributed on up to 8 nodes, each with 28 Intel CPU cores and 6 GPUs (K80). While running the workflows, PLLib captures data at runtime and sends them to ProvTracker which in turn sends them to ProvManager service deployed externally on the virtual Kubernetes cluster, which finally stores them in the Prov DBMS. A provenance capture overhead analysis of ProvLake using synthetic workloads to highly stress the system and comparison with a competing system has been presented in a previous work [17]. Here, we first present a performance analysis testing different settings for provenance analysis, and then a scalability analysis, both using real ML workloads. We measure the overall execution time of the training workflow script, repeating each test at least 10 times and we plot the boxplots of the repetitions and the numeric values used in-text refer to the median of the repetitions.

Experiment 1: varying provenance capture settings. For a baseline, we first execute the training without any provenance capture, then we vary the queue size in PLLib (*i.e.*, amount of provenance capture requests accumulated in PLLib), diskless vs. diskful (*i.e.*, saving or not provenance data in a log file on disk), and online vs. offline (*i.e.*, storing or not provenance data in the DBMS, available for online provenance queries during the execution). As for the training datasets, we use a curated and labeled real seismic dataset using a specific range of seismic slices (corresponding to a regional section of a seismic cube) defined by the model trainer. The results are in Fig. 5, where the fastest result is for Queue Size = 50,

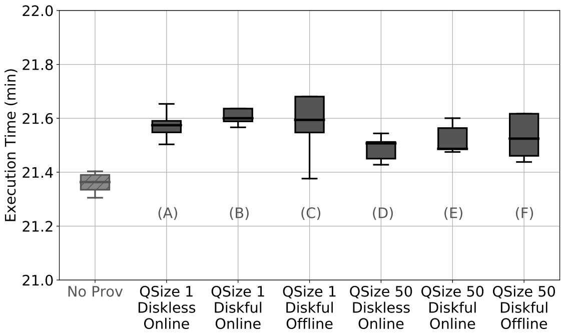


Fig. 5: Varying prov. capture. Setting D adds 0.67% overhead.

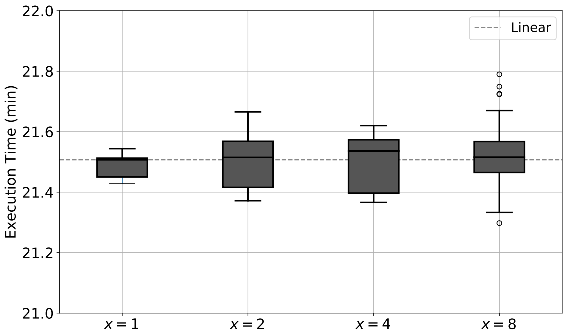


Fig. 6: Weak scalability analysis.

Diskless, Online (Setting D). Comparing with the setting with no provenance capture, the added execution overhead in this case is only 8.6 seconds on top of 21.3 minutes, *i.e.*, 0.67%, which is considered negligible.

To analyze the queue size, we compare Settings A–C with D–F and we see larger queues provide faster provenance capture since there is less but larger communication with ProvTracker service. For instance, Setting A is about 7% slower than D. However, very large queues have drawbacks as they introduce higher latency between the event being captured in the workflow execution and the provenance record being stored in the database, caused by the retention of provenance capture events in PLLib's queue. Nevertheless, for the settings with queue size 50 (D–F), a latency of less than 5 seconds between the actual occurrence of the event and its provenance being registered in the database, available for queries, can be considered near real-time and good enough even for training monitoring. To analyze diskless vs. diskful settings, we compare Setting A with B and C; and D with E and F. Diskless is faster than diskful, as the latter introduces more I/O operations at runtime. However, comparing only the medians, the difference is negligible (less than 0.1%). Thus, because of a higher fault-tolerance provided by a diskful setting, it may be interesting to append provenance data onto a file on disk, locally in the cluster where the workflow runs. Similarly, comparing the medians, we observe that the difference between online vs. offline (*e.g.*, setting B vs. C or E vs. F) is also small, about 1%. Therefore, despite (D) being the fastest setting, (E) may be preferred because its performance is nearly the same as (D) and it has the advantage of backup storage for provenance data, which is quite important as provenance is used for quality assessment and reproducibility.

Experiment 2: scalability analysis. In this experiment, we want to confirm if the execution strategies on an HPC cluster are keeping the overhead low in a real ML workload, running multiple training workflows in parallel. We run a weak scalability test by increasing the number of processing units while increasing the data size. We use the fastest setting of the previous experiment (*i.e.*, D) and the same seismic cube. To set up the training datasets, the trainer selects up to 8 different sets of seismic slices, where each set has the same length (*i.e.*, nearly the same data size). Thus, for $x \in \{1, 2, 4, 8\}$, there are x workflows running on x nodes in parallel, summing $28x$ Intel CPU cores, $6x$ GPUs, $4992 * 6x$ CUDA GPU cores, using in total an input dataset with size $x * datasize$, where $datasize$ is the size of a dataset formed by 1 set of seismic slices. The results are in Fig. 6, where we illustrate the linear scalability as a horizontal line passing through the median of the smallest setting ($x = 1$). Ideally, the medians should be near this line. If they are not, it means that ProvTracker is taking too long to answer, caused by high stress in the system due to too many provenance capture requests, adding latency to the training. However, we see that even in the largest setting (*i.e.*, $x = 8$), the execution time remains close to the linear curve. The boxes remain within a small margin of 0.2 min (or 0.9% of the $x = 1$ median) between 21.4 and 21.6 min, meaning that the system delivers a constant and predictable behavior even at larger scales. We note though that the variance grows with the scale, caused by the larger number of parallel tasks. Therefore, we conclude that at least for this scale (up to 48 K80 GPUs), the provenance capture system delivers good scalability.

B. Use Case Validation

We explain how ProvLake supports queries Q1–Q6 in the O&G use case, illustrated in Fig. 7 and described in Section II-C. As shown in the figure, the phases of the ML lifecycle in CSE are interconnected, as data generated in a phase are consumed in another. Essentially, ProvLake tracks and maintains such interconnections in a provenance data graph as millions of RDF triples (about 30M in total after all models have been trained in this use case) as the chained data transformations in the multiple, distributed workflows composing the inner phases of the major phases of the lifecycle run. The data in the figure are represented as RDF resources, *i.e.*, instances that extend `prov:Entity` and PROV-ML specializations. Each of these instances receives a URI, which works as a global identifier throughout the lifecycle. Each trained model generated in the learning phase is represented as an RDF resource, as well as the model hyperparameters of each trained model, the evaluation metrics, and a reference (file path) to the actual model file stored in the file system. Execution data, such as file system metadata, cluster's hostname and node names used in the HPC jobs, job ids in the cluster scheduler, and start and end timestamps of each block of provenance capture events are associated to the trained models in the provenance data graph.

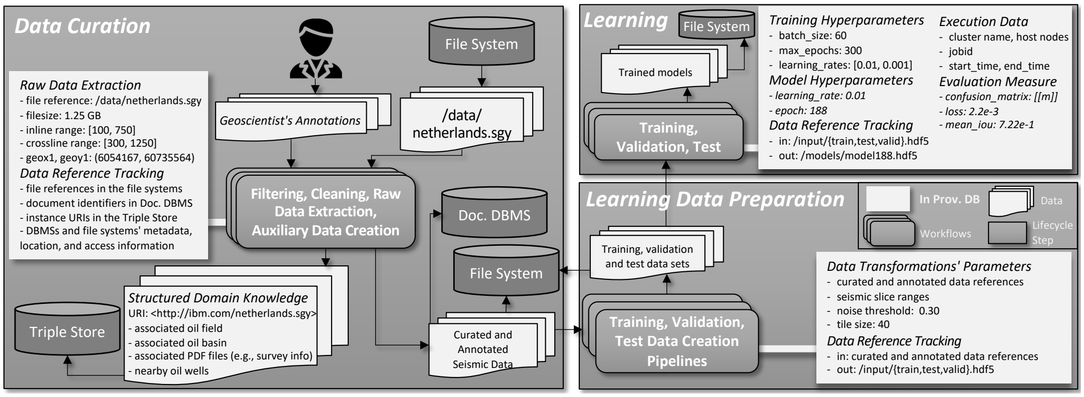

Data Curation
File System
Raw Data Extraction
- file reference: /data/netherlands.sgy
- filesize: 1.25 GB
- inline range: [100, 750]
- crossline range: [300, 1250]
- geox1, geoy1: (6054167, 60735564)
Data Reference Tracking
- file references in the file systems
- document identifiers in Doc. DBMS
- instance URIs in the Triple Store
- DBMSs and file systems' metadata, location, and access information
Geoscientist's Annotations
/data/netherlands.sgy
Filtering, Cleaning, Raw Data Extraction, Auxiliary Data Creation
Doc. DBMS
File System
Triple Store
Structured Domain Knowledge
URI: <http://ibm.com/netherlands.sgy>
- associated oil field
- associated oil basin
- associated PDF files (e.g., survey info)
- nearby oil wells
Curated and Annotated Seismic Data
Learning
File System
Trained models
Training, Validation, Test
Training Hyperparameters
- batch_size: 60
- max_epochs: 300
- learning_rates: [0.01, 0.001]
Model Hyperparameters
- learning_rate: 0.01
- epoch: 188
Data Reference Tracking
- in: /input/{train,test,valid}.hdf5
- out: /models/model188.hdf5
Execution Data
- cluster name, host nodes
- jobid
- start_time, end_time
Evaluation Measure
- confusion_matrix: [[m]]
- loss: 2.2e-3
- mean_iou: 7.22e-1
Learning Data Preparation
In Prov. DB
Data
Workflows
Lifecycle Step
Training, validation and test data sets
Training, Validation, Test Data Creation Pipelines
Data Transformations' Parameters
- curated and annotated data references
- seismic slice ranges
- noise threshold: 0.30
- tile size: 40
Data Reference Tracking
- in: curated and annotated data references
- out: /input/{train,test,valid}.hdf5

Fig. 7: Provenance data tracking in an O&G use case for the ML lifecycle in CSE.

Similarly, in the learning data preparation phase, there are several data transformations in a data pipeline that transform the curated and annotated scientific data into training, validation, and test datasets. Each data transformation is parameterized. Parameters specify, for instance, noise filter thresholds, size (in pixels) of tiles which will serve a seismic image classifier, ranges of regions (called seismic slices, *e.g.*, inline and crossline slices) of the seismic cube that will form the training dataset. Each value of these parameters, the name of the transformation, execution data, and data references to input and output data physically stored in the data stores are captured and represented in ProvLake's provenance data graph. For the data curation phase, ProvLake captures provenance when the data-intensive scripts that clean, filter, and create auxiliary data run. When processing raw scientific files, important data which will help to answer the queries are extracted, such as geographic coordinates embedded as metadata in raw SEG-Y seismic files (represented as a `netherlands.sgy` in the figure), associated to the file's URI, and stored in the provenance database. Yet, geoscientists input important annotations into some of those scripts including associated oil fields, basins, oil wells, and pieces of texts from PDF documents with survey information related to the geological data acquisition process. These annotations are stored in a domain-specific database, externally to the provenance database, stored in a Triple Store. In this case, ProvLake's ability to keep track of data distributed in multiple stores helps to maintain the data relationships between the raw files in the file system and the structured knowledge stored in another database. Auxiliary data, such as polygons of the seismic cube are stored in the Document DBMS, and the data references are similarly tracked and related to the raw files. Other data, such as implementation details, software name and version, are captured and stored in the provenance database, following the PROV-ML, but, for simplicity, we do not show them in the figure. As the data and their relationships are properly tracked while the workflows execute, ProvLake enables answering online, offline, intra- and inter-training provenance queries to analyze ML data, domain-specific data, and execution data throughout the phases of the lifecycle, exemplified by the queries Q1–Q6.

To submit queries, the user sends a GET or POST request to one of PolyProvQueryEngine's endpoints. Then, PolyProvQueryEngine sends requests to ProvManager. Most of the queries are answered with simple graph traversals using standard SPARQL features. For instance, to answer Q1, the user provides a trained model URI (generated in the learning phase) and the query should traverse in the provenance data graph backward until the raw seismic file's URI (processed in the data curation phase). To return the geographic coordinates and number of seismic slices, the query uses the extracted data related to the seismic file. To return the oil basin and oil field information, the query retrieves data from the resource, in the Triple Store, that represents structured knowledge about the seismic file. For Q2 and Q6, similar graph traversal is executed. Other queries require analytical operators, such as Q3, which requires finding the trained model with least (using `min()` native SPARQL operator) loss, and returning its hyperparameters. Q4 and Q5 make use of execution data to provide basic statistics (`min(), max(), avg()` operators) about the execution time of training iterations.

VI. Related Work

Some works have addressed provenance tracking in the ML context [9–11, 24]. However, they are mainly focused on the learning and learning data preparation phases, failing to trace back from the trained models until the raw domain-specific data curated in workflows in the curation phase. These solutions often come with one single system to manage execution, data, and provenance of the whole lifecycle, but in order to do so, users need to develop their workflows in such a system. Although it is a good fit for simple projects (*e.g.*, the same user designs ML models, curates, prepares the data and trains the ML models), it is not for CSE, which is considerably more complex and heterogeneous. It is unrealistic to expect that all phases, their execution, and the processed data will be managed by one single system. Alternatively, provenance tracking systems [12–16] can be coupled to a CSE workflow, providing provenance support while not significantly changing the way CSE users develop their applications. However, these solutions fail to track the interconnections between workflows and fail to track data processed in multiple heterogeneous stores. Also, some of them [12] add high provenance capture overhead, preventing their adoption in CSE. Finally, none of them has a provenance data representation capable of representing ML-specific and domain-specific data, as we propose with PROV-ML, with extensions of W3C PROV [18] and MLS [22].

On new provenance data representations for ML, some works addressed the gap between the experiments of a ML workflow execution and a standard representation to provide reproducible experiments [19, 22, 25]. Esteves *et al.* [25] introduce W3C PROV-compliant provenance in these workflows in ML. They provide a machine-readable vocabulary and a common schema for reproducibility in various frameworks and workflow systems. However, it lacks details of the ML phases itself. Publio *et al.* [19] present a new ML data representation based on MEX vocabulary [25] to improve processes on ML workflows. Nonetheless, they lack a explicit separation between prospective and retrospective provenance, limiting provenance data understanding. Moreover, these works are focused on the learning phases of the lifecycle, whereas the interconnections with workflows in prior phases, like for data curation, are not provided. Finally, none of these solutions has a provenance tracking system as we are proposing.

VII. Conclusions

In this work, we addressed the problem of tracking the data transformations in the ML lifecycle, focusing on CSE. We showed that heterogeneity in several dimensions, including different human expertise, workflows, data stores, execution machines, among others, adds a significant complexity that must be addressed to support provenance tracking in the ML lifecycle; end-to-end from raw scientific data files to trained models. To the best of our knowledge, this is the first work that characterizes provenance as an essential aspect to be managed for the track of data in the ML lifecycle in CSE.

Although existing provenance tracking solutions that can be coupled with workflows contribute with the flexibility needed in CSE projects, they fail to support the heterogeneous nature of the lifecycle. After the practical experience of extending ProvLake for the lifecycle, we draw the following lessons:

(i) The characterization of provenance in the lifecycle allows for an understanding of the different needs of different persona as it drives the provenance tracking to answer key online and offline, intra- and inter-training provenance queries capable of analyzing, in an integrated way, ML data, domain-specific data, and execution data, throughout the data curation, data preparation and learning phases of the lifecycle. We observed that the data curation step, which is often neglected by ML systems, is the most complex part in CSE and needs to be addressed carefully for provenance analysis.

(ii) In CSE, it is necessary to integrate provenance from multiple workflows that process domain-specific data in the data curation phase and ML data in the learning phases of the ML lifecycle; otherwise, important data are not tracked properly. In practice, this is often done manually, which is time consuming and error prone. To achieve this integration, it was key to create a representation that leverages ML and domain-specific data. Therefore, we created PROV-ML, which is compliant to W3C definitions, namely PROV and ML Schema. We hope such representation can be adopted by other systems in this area.

(iii) Architectural design decisions, such as a microservice architecture and a lightweight provenance capture library (with less than 1% of overhead), are essential for efficient tracking with comprehensive provenance analysis. We observed this through a real O&G case running on a testbed of 48 GPUs.

Acknowledgment

We thank Marcelo Costalonga and Daniela Szwarcman from IBM Research Brazil for their help, and the WORKS19 anonymous reviewers for their valuable feedback. This work was partially funded by CNPq, FAPERJ, and Inria Associated Team SciDISC.

References

[1] J. Hesthaven and G. Karniadakis. (2019) Scientific machine learning workshop. https://icerm.brown.edu/events/ht19-1-sml

[2] Y. Gil, S. A. Pierce, H. Babaie *et al.*, “Intelligent systems for geosciences: an essential research agenda,” *CACM*, 2018.

[3] M. Raissi, P. Perdikaris, and G. Karniadakis, “Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *J. Comp. Physics*, 2019.

[4] E. Rodrigues, I. Oliveira, R. Cunha *et al.*, “DeepDownscale: a deep learning strategy for high-resolution weather forecast,” in *IEEE eScience*, 2018.

[5] D. S. Chevitarese, D. Szwarcman, E. V. Brazil *et al.*, “Efficient classification of seismic textures,” in *IJCNN*, 2018.

[6] N. Polyzotis, S. Roy, S. Whang *et al.*, “Data lifecycle challenges in production machine learning: a survey,” *SIGMOD Rec.*, 2018.

[7] E. Deelman, A. Mandal, M. Jiang *et al.*, “The role of machine learning in scientific workflows,” *Int. J. HPC*, 2019.

[8] M. Herschel, R. Diestelkmper, and H. Ben Lahmar, “A survey on provenance: What for? what form? what from?” *VLDB J.*, 2017.

[9] H. Miao, A. Li, L. S. Davis *et al.*, “Towards unified data and lifecycle management for deep learning,” in *ICDE*, 2017.

[10] Z. Zhang, E. R. Sparks, and M. J. Franklin, “Diagnosing machine learning pipelines with fine-grained lineage,” in *HPDC*, 2017.

[11] M. Zaharia, A. Chen, A. Davidson *et al.*, “Accelerating the machine learning lifecycle with MLflow,” in *IEEE Data Eng. Bulletin*, 2018.

[12] I. Suriarachchi, S. Withana, and B. Plale, “Big provenance stream processing for data intensive computations,” in *IEEE eScience*, 2018.

[13] L. Carvalho, K. Belhajjame, and C. Medeiros, “A PROV-compliant approach to script-to-workflow process,” *The Sem. Web J.*, 2018.

[14] J. Pimentel, J. Freire, L. Murta *et al.*, “A survey on collecting, managing, and analyzing provenance from scripts,” *ACM Surv.*, 2019.

[15] V. Silva, D. de Oliveira, P. Valduriez *et al.*, “DfAnalyzer: runtime dataflow analysis of scientific applications using provenance,” *PVLDB*, 2018.

[16] V. Silva, R. Souza, J. Camata *et al.*, “Capturing provenance for runtime data analysis in computational science and engineering applications,” in *IPAW*, 2018.

[17] R. Souza, L. Azevedo, R. Thiago *et al.*, “Efficient runtime capture of multiworkflow data using provenance,” in *IEEE eScience*, 2019.

[18] L. Moreau and P. Missier. (2013) PROV-DM: the PROV data model. https://www.w3.org/TR/prov-dm/

[19] G. C. Publio, D. Esteves, A. Ławrynowicz *et al.*, “ML Schema: exposing the semantics of machine learning with schemas and ontologies,” in *Reproducibility in ML@ICML*, 2018.

[20] R. Souza, V. Silva, A. Coutinho *et al.*, “Data reduction in scientific workflows using provenance monitoring and user steering,” *FGCS*, 2017.

[21] R. Souza, V. Silva, J. J. Camata *et al.*, “Keeping track of user steering actions in dynamic workflows,” *FGCS*, 2019.

[22] A. awrynowicz *et al.* (2019) Machine learning schema community group. https://www.w3.org/community/ml-schema/

[23] Provlake website. https://ibm.biz/provlake

[24] A. Kumar, R. McCann, J. Naughton *et al.*, “Model selection management systems: the next frontier of advanced analytics,” *SIGMOD Rec.*, 2016.

[25] D. Esteves, D. Moussallem, C. B. Neto *et al.*, “MEX vocabulary: a lightweight interchange format for machine learning experiments,” in *SEMANTICS*, 2015.