Automated Documentation of End-to-End Experiments in Data Science

Sergey Redyuk
Technische Universität Berlin, Germany
sergey.redyuk@tu-berlin.de

***Abstract*—Reproducibility plays a crucial role in experimentation. However, the modern research ecosystem and the underlying frameworks are constantly evolving and thereby making it extremely difficult to reliably reproduce scientific artifacts such as data, algorithms, trained models and visualizations. We therefore aim to design a novel system for *assisting data scientists with rigorous end-to-end documentation of data-oriented experiments*. Capturing data lineage, metadata, and other artifacts helps reproducing and sharing experimental results. We summarize this challenge as *automated documentation* of data science experiments. We aim at reducing manual overhead for experimenting researchers, and intend to create a novel approach in dataflow and metadata tracking based on the analysis of the experiment source code. The envisioned system will accelerate the research process in general, and enable capturing fine-grained meta information by deriving a declarative representation of data science experiments.**

***Keywords*-experimentation in data science; reproducibility; automatic tracking of metadata; responsible data management**

I. Introduction

Reproducibility lays at the core of the scientific method. Publically accessible, reusable artifacts drive science by enabling researchers to build upon existing knowledge, to compose it and to transfer it across domains. With the growing number of data science (DS) applications in almost every field of research, a burst of a new sort of artifacts - data, software, interactive demos - makes sharing and reuse of scientific contributions challenging. On the one hand, this is due the absence of unified standards for designing end-to-end data science workflows. On the other hand, there is a constant and rapid development of new analytical solutions [1]–[3], such as the emergence of deep learning engines, which enriches the range of tools available in research, yet *makes scientific artifacts extremely hard to compare*. For instance, two pieces of source code for data loading written with low-level built-in functions and high-level packages may be semantically equivalent, but it will be difficult to recognize this due to syntactic differences. An additional challenge is to support collaborative efforts of research teams, by providing access to the artifacts for further reuse across teams and domains.

Modern data-oriented experiments are often conducted by following a "Try-Fail-Learn-Iterate" paradigm: scientists focus on fast, iterative exploration and hypothesis testing, which allows them to quickly understand which ideas work. Often the need for capturing this dynamic process of experimenting and prototyping is ignored. As a result, by the time a data science prototype evolves and demonstrates promising outcomes, researchers often realize that they cannot reproduce the experiment, due to the absence of a 'lab notebook' that contains the experiment documentation. Similar problems arise in industry as well [2], [4].

Observing the aforementioned challenges, we conclude that automated documentation of experiments is a crucial requirement for modern research environments. We denote documentation of data science experiments as the process of recording all the data that are necessary to achieve full reproducibility of the experiment (source code versioning, end-to-end tracking of data provenance, environment settings, meta information, specifications of stochastic processes during the data science experiment, etc.). The high complexity of this task makes it time-consuming and tedious to perform manually. Thus, we formulate the goal for this Ph.D. project as *streamlining the process of end-to-end tracking of data lineage and meta information, inspecting the artifacts produced during data science experiments, as well as sharing, and reusing these results across teams and domains*.

There are two major approaches to promote research reproducibility and encourage experiment documentation among scientists. The first approach focuses on the extrinsic motivation of researchers. For example, the ACM created a policy of artifact review and badging to help SIGs recognize and acknowledge reproducibility in published research [5]. The second approach focuses on the technical side. It aims at providing libraries and systems for tracking and reproducing experiments [3], [6]. Although the latter is of great value, current solutions such as MLflow or Amazon's solution [4] are often either problem- or package-specific (Section III), and require manual specification of the metadata to track. To mitigate this drawback, we envision to design a system that collects metadata and data lineage automatically, and records experiments in a way that imposes minimal overhead on the experimenting user.

We aim to accelerate research in data science and shorten the innovation cycle for newly developed artifacts. We focus on two main problems: (i) how to automate the tracking of metadata, lineage, and other intermediate results of the experiments without introducing significant overhead (w.r.t. the total execution time) or requiring scientists to alter the source code of their experiments; and (ii) how to derive

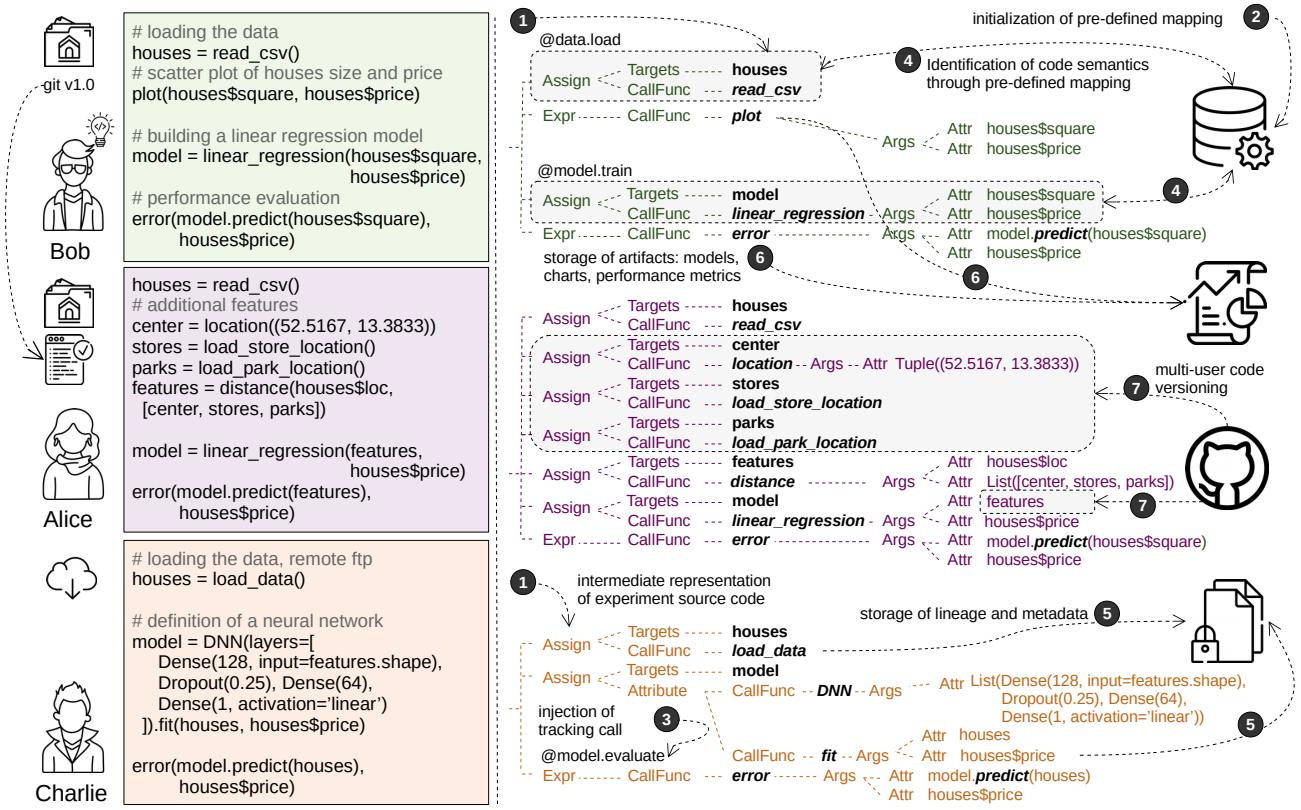

Figure 1. Overview of how the envisioned system would address our example scenario. For all DS experiments conducted by the involved individuals, the system analyses the source code, detects patterns in the intermediate representation and identifies their semantics via a predefined mapping for known DS frameworks. Next, our systems decomposes the source code into logical segments in accordance with the data science task, and automatically injects tracking code, which persists meta information, data lineage and other artifacts at runtime.

a declarative representation of the experiment (a detailed specification, along the lines of the W3C ML Schema initiative [7]) from arbitrary source code to enable capturing fine-grained metadata. We envision the following key contributions of the Ph.D. project:

(i) Automated injection of metadata and lineage tracking calls into DS experiments at runtime via source code analysis;
(ii) Extraction of a declarative intermediate representation of a data science experiment that identifies phases such as data preprocessing or model selection;
(iii) Design and population of an experiment database to enable search and reuse capabilities over previous experiments and their declarative representation.

These contributions potentially have high impact for several research areas. The first area is *responsible data management* [8]. Using data responsibly means that the decisions people make based on data analysis have to comply to ethical norms and legal regulations. Our proposed solution addresses several key points of responsible data management: transparency, accountability and tracking. By recording the exact pathway the model takes to make predictions, we will enable retrospective analysis of these pathways to understand how a particular decision was made, how to reduce bias, and how to make "black-box" machine learning (ML) models more transparent. In settings where legal obligations bind researchers or ML engineers, we facilitate presenting evidence whether decisions made by the ML model have been fair. The second area with potential impact is the *automation of machine learning*. By recording the dynamic process of trials and errors made during the experiments, we are able to gain insights on how experimenting in data science works in general, and design novel optimizations for end-to-end DS workflows.

II. Proposed Work

We introduce an example scenario for our envisioned system and discuss the automatic injection of metadata and lineage tracking code into experiments (Figure 1, left).

A. Example Scenario

Imagine Bob who wants to predict the price of an apartment based on its characteristics. He creates a `python` script, loads the dataset from the local filesystem, and conducts exploratory data analysis. He visualizes how house prices correlate with the features that he extracted (size, location, etc.). Next, Bob normalizes the data and builds a linear regression model to test whether the price depends on the size of the apartment. Performance analysis shows

that the model miscalculates the price by 20,000 euros. He shares this result with his colleague Alice. She suggests to use the house location in order to compute distances to the city center, to nearest stops for public transportation, supermarkets, parks, or schools, and to leverage these features to improve the model's accuracy. She takes Bob's code, adds the new features, retrains the model, checks whether the performance has increased, and finds a lower error of only 10,000 euros of misprediction. Bob and Alice then go to Charlie, a fan of deep learning, who suggests to take the original dataset and train a deep neural network model that learns data representation by itself. After several changes in the model architecture and hyperparameters, Charlie creates a neural network that mispredicts the price by 5,000 euros.

After prototyping, Bob, Alice and Charlie become curious. They wonder which factors lead to significant boosts in model performance and which changes only slightly improved the accuracy. Alice and Bob compare their code in a straightforward way - the source code version control system detects the exact changes Alice made; linear regression by design has an optimal solution that can be easily recomputed. Later they realize that (1) the data Charlie got from the remote source differs from Bob's dataset; (2) Charlie did not save the model and therefore cannot reproduce the predictions of the neural network reliably; (3) they cannot determine which change had been more important - new data features or the complex model. The system that we propose is aimed to provide the basis for resolving the problems that Bob, Alice and Charlie have faced.

B. Automated Documentation in the Example Scenario

If Bob, Alice and Charlie had executed their experiments with the assistance of our envisioned tracking system, several additional steps would occur in the source code compilation phase (as illustrated in the right part of Figure 1). After parsing the source code (left), the system obtains a tree-based representation that enables querying over object and variable names (houses, model), function signatures, numerical values (Assign, Expr, FuncCall) etc. ❶. We use this stringent intermediate representation to detect patterns in subtrees, match them with a predefined mapping to existing machine learning libraries (e.g., scikit-learn), and annotate the corresponding parts in the code. This mapping contains information about public packages for data analysis (such as the `LinearRegression` model from `python`'s `sklearn` package) and the corresponding DS task that each particular class intends to solve (i.e., regression) ❷. Other DS tasks are defined by standard process models in data science [9] and include data loading, cleaning, integration, model training, selection and evaluation, communication of the results etc. The annotations are then used to *inject calls to tracking code* into the experiment code, which capture and persist runtime changes of selected variables ❸. Next, we *decompose the code into logical chunks* based on the DS task ❹. Automated source code decomposition is one of the envisioned key contributions of the project that (i) allows for a high-level intermediate representation of the data science process to be encoded in the experiment, and (ii) captures its logical intermediate results - checkpoints at the end of each component - for further reuse ❺. A high-level representation stored in an experiment database ❻ enables advanced querying based on the logical structure of the experiment. This, in turn, brings new opportunities for meta-learning and workflow optimization. Furthermore, it enables automatic restoration of the intermediate results which accelerates the innovation cycle. As experimenting is an iterative process, multi-user source code versioning ❼ helps to track what exact changes in the code affect the model performance.

Based on the analysis of the source code and its knowledge of the semantics of existing DS software, the system is able to extract numerical constants (potential hyperparameters) and to choose which artifacts to track. It can also invoke package-agnostic calls to infer implicit values (such as default values that are not present in the code yet affect the model), and inject tracking calls by altering the source code in order to "track" the changes at runtime. After the analysis is completed, the compiler translates the updated syntax tree into a bytecode representation for further execution. Besides the experiment itself, the system collects information about the operating system, hardware specification, and the experiment's software dependencies. It automatically keeps records on the whole experiment ecosystem and detects changes that immediately affect the experiment's meta information. We can later analyze this information to determine which factors influence our experiment (in our scenario, to help Bob, Alice and Charlie to systematically compare their solutions).

C. Challenges

A major challenge will be to enable the system to handle scenarios where the semantics of the experiments are difficult to infer automatically. We give an example of such a case in Listing 1. This script loads and scales the data, trains a logistic regression model, and evaluates its performance without using any high-level packages. Hence, we cannot leverage our existing knowledge about data science software APIs. For this case, we aim to (i) analyze experiment databases to detect frequent patterns that describe each DS phase; and (ii) evaluate the control flow of the experiment to detect chains of variable transformations and extract chunks of code that transform a single placeholder (i.e. the low-level implementation of data scaling which slices the dataset, computes means and standard deviations column-wise without altering any other variables).

In most of the cases, the experiment source code is a combination of both calls to the APIs of high-level packages and low-level imperative statements. For instance, eager execution mode of the `tensorflow` package might combine a

high-level symbolic definition of a neural network and low-level declaration of the optimization loop. This approach enables enhanced debugging capabilities for developers of neural networks, but makes it more challenging to identify the appropriate places to inject the tracking code.

```
# gradient descent learning algorithm
def grad_desc(X, Y, alpha, iterations):
  W = zeros((X.shape[1], Y.shape[1]))
  for i in  xrange(iterations):
    W -= alpha * d_cost(X, W, Y)
  return W

# loading the input dataset
X_train, y_train, X_test, y_test = load(...)
# manual standard scaling of the data
X_train = zero_mean(X_train) / std(X_train)
X_test = zero_mean(X_test) / std(X_test)

W = grad_desc(X_train, y_train, 0.1, 900)
# measurement of prediction error
mean( abs(sigmoid(dot(X, W)) - Y))
```

Listing 1. Direct implementation of logistic regression without reliance on high-level packages, such as scikit-learn. In this case, it becomes difficult to understand the semantic structure of the experiments as our system cannot rely on the knowledge regarding existing libraries.

D. Evaluation

A difficult task in this line of research is to measure the performance of the envisioned system. We aim to evaluate the system in two different types of data science experiments: (i) on manually created examples, where the information about the logical structure of the experiment, ML models, hyperparameters, ecosystem is available for comparison as ground truth, and (ii) on automatically collected examples from public repositories, where the ground truth can only partially be inferred (e.g., by taking the logical decomposition of `jupyter` notebooks into account). We plan to measure the number of derived DS components and their classification, to evaluate if the system succeeds in decomposition. We start with a simple binary comparison whether the derived structure of the experiment matches the ground truth, and then develop a more elaborate similarity score depending on the experimentation scenario. We intend to compare the intermediate artifacts (lineage, ML models, performance metrics) of both the original version of the experiment and a version recreated based on the captured metadata, to check whether the system achieves reproducible results. We aim to apply similarity metrics to each artifact computed during the experiment in order to assign a cumulative similarity score of how well the experiment is reproduced. Furthermore, we will measure the overhead in execution time induced by the system on these examples.

E. Implementation

We intend to implement the proposed solution for `python`, a programming language of choice in this project, since it is widely used among researchers and data scientists for experimenting and prototyping, has low entry barriers, and contains an extensive list of packages such as `NumPy`, `pandas`, `tensorflow` or `sklearn` which facilitate the development of data analysis artifacts. We limit our scope by considering use cases which have *tabular data* as an input. Algorithms for the source code analysis are, in most cases, language- and implementation-dependent. Thus, we choose the language version 3.7.0 and `CPython` implementation as the mainstream branch that contains all the edge features of the language, and leave alternative `PyPy` implementations that support JIT optimization for future work. Moreover, we start with classical sequential data science pipelines and tackle complex production pipelines (e.g., that involve multiple data sources or models) later on.

III. Related Work

Academic and industrial research on experiment reproducibility includes many areas, such as the design of experiment databases [6], [10] and digital libraries [11], systems for tracking of metadata and provenance [4], code versioning (`git` and `dvc`), packaging the artifacts and sharing the results (`docker` and `quilt`). We briefly outline several existing systems that are related to the project. In the fiels of ML model diagnosis, MISTIQUE [3] is a tool that is capable of storing intermediate results of the experiment, and deciding whether the artifacts should be materialized or recomputed on demand. It inherits the goals of ModelDB [12] - a database for long-lived publicly accessible models in computational neuroscience. In the area of the end-to-end experiments lifecycle management, Comet, MLflow, and the system developed at Amazon [4], [13] are platforms that allow tracking and packaging of the experiments, as well as extracting, storing and managing metadata and provenance of common ML artifacts. In comparison to these systems, our solution is package-agnostic and allows for full automation of experiment documentation without requiring the user to alter the code or specify the parameters to track. Collaborative open science platforms, such as OpenML [6] and Kipoi [10], provide simple access to stored machine learning data, models, pipelines and experimental results. These systems address the problem of reproducibility and sharing but do not take the specification of experiments, metadata tracking, or high-level workflow decomposition into account. When it comes to packaging of the scientific artifacts, ReproZip [14] enables automatic collection of experiments (by tracing system calls) in conjunction with all required data files, libraries, and environment variables. Our solution extends the functionality of ReproZip by collecting a declarative specification of the experiment, its data provenance and meta information. noWorkflow [15] is an open-source tool for collecting data provenance from Python scripts, including data about the script execution and versioning. It also uses methods of source code analysis such as the analysis of AST and bytecode. Its key advantage is to

capture data provenance even if no workflow systems are applied, whereas the goal of our solution is to infer the workflow and use it for the source code decomposition.

IV. Future Directions

Lastly, we outline the expected directions for the final years of the Ph.D. project. Since collecting and storing artifacts in all their intermediate states requires excessive volumes of memory, we need effective compression mechanisms. Examples of these mechanisms are delta encoding for storing various versions of the same artifact, serialization for compressing the data, hybrid approaches for experiment databases - in order to achieve optimal performance for each task (storage mechanism, message passing interfaces, handling time-series data etc.). An alternative direction is related to the area of meta-learning. Capturing additional information about data-oriented experiments (as dynamic processes with all the intermediate steps being recorded) will bring new insights on how scientists construct data science pipelines, lead to new algorithms for automated machine learning, and accelerate the innovation cycle. Ultimately, the proposed solution might lead to a fully automated end-to-end management system for DS experiments.

V. Conclusion

Reproducibility plays an important role in research and ML application development. Artifacts that can be verified by a third party, ported to another environment, and reused in the future accelerate the innovation cycle. Although the core benefits of designing reproducible artifacts are agreed upon, achieving reproducibility in modern R&D environments is a complicated task that brings several challenges.

We focus on the challenge of automated lineage and metadata tracking for data-oriented experiments. We proposed a system for *automated injection of metadata and lineage tracking calls* into DS experiments via source code analysis, and thereby extract a *declarative representation* of the experiments. The captured meta information will facilitate reproducibility, a fine-grained search over the stored experiment sessions, and their further reuse. Thus, we envision the area of meta-learning and automation of end-to-end experiments management systems as promising future directions.

We would like to acknowledge the support of the Helmholtz Einstein International Berlin Research School in Data Science (HEIBRiDS), Max Delbrück Center for Molecular Medicine (MDC), TU Berlin, and the Berlin Center for Machine Learning (BZML) 01IS18037A. We thank Dr. Sebastian Schelter (NYU) and Prof. Dr. Volker Markl (TU Berlin) for guidance and valuable contributions.

References

[1] E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht, "KeyStoneML: Optimizing pipelines for large-scale advanced analytics," in *ICDE*, 2017, pp. 535–546.

[2] D. Baylor, E. Breck, H. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, C. Y. Koo, L. Lew, C. Mewald, A. N. Modi, N. Polyzotis, S. Ramesh, S. Roy, S. E. Whang, M. Wicke, J. Wilkiewicz, X. Zhang, and M. Zinkevich, "TFX: A tensorflow-based production-scale machine learning platform," *KDD*, pp. 1387–1395, 2017.

[3] M. Vartak, J. M. F. da Trindade, S. Madden, and M. Zaharia, "MISTIQUE: A System to Store and Query Model Intermediates for Model Diagnosis," *SIGMOD*, pp. 1285–1300, 2018.

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

[5] N. Ferro and D. Kelly, "SIGIR initiative to implement ACM artifact review and badging," in *ACM SIGIR Forum*, vol. 52, no. 1, 2018, pp. 4–10.

[6] J. N. Van Rijn, B. Bischl, L. Torgo, B. Gao, V. Umaashankar, S. Fischer, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren, "OpenML: A collaborative science platform," *ECML-PKDD*, pp. 645–649, 2013.

[7] D. Esteves, A. Lawrynowicz, P. Panov, L. Soldatova, T. Soru, and J. Vanschoren, "ML Schema Core Specification," W3C, http://www. w3. org/2016/10/mls, Tech. Rep., 2016.

[8] S. Abiteboul, G. Miklau, J. Stoyanovich, and G. Weikum, "Data, Responsibly," *Dagstuhl Reports, Seminar 16291*, vol. 6, no. 7, pp. 42–71, 2016. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2016/6764

[9] A. I. R. L. Azevedo and M. F. Santos, "KDD, SEMMA and CRISP-DM: a parallel overview," *IADS-DM*, 2008.

[10] Z. Avsec, R. Kreuzhuber, J. Israeli, N. Xu, J. Cheng, A. Shrikumar, A. Banerjee, D. S. Kim, L. Urban, A. Kundaje, O. Stegle, and J. Gagneur, "Kipoi: accelerating the community exchange and reuse of predictive models for genomics," *bioRxiv*, 2018. [Online]. Available: https://www.biorxiv.org/content/early/2018/07/24/375345

[11] R. Palma, P. Hołubowicz, O. Corcho, J. M. Gómez-Pérez, and C. Mazurek, "Rohub—a digital library of research objects supporting scientists towards reproducible science," *Semantic Web Evaluation Challenge*, pp. 77–82, 2014.

[12] M. L. Hines, T. Morse, M. Migliore, N. T. Carnevale, and G. M. Shepherd, "Modeldb: a database to support computational neuroscience," *Journal of computational neuroscience*, vol. 17, no. 1, pp. 7–11, 2004.

[13] S. Schelter, J.-H. Böse, J. Kirschnick, T. Klein, and S. Seufert, "Declarative Metadata Management: A Missing Piece in End-To-End Machine Learning," 2018.

[14] F. Chirigati, R. Rampin, D. Shasha, and J. Freire, "Reprozip: Computational reproducibility with ease," ser. SIGMOD, 2016, pp. 2085–2088.

[15] J. F. Pimentel, L. G. P. Murta, V. Braganholo, and J. Freire, "noWorkflow: a Tool for Collecting, Analyzing, and Managing Provenance from Python Scripts," in *PVLDB*, vol. 10, 2017, p. 12.