




Towards Tracking Provenance from Machine Learning Notebooks

Dominik Kerzel¹^a, Sheeba Samuel^{1,2}^b and Birgitta König-Ries^{1,2}^c

¹Heinz Nixdorf Chair for Distributed Information Systems, Friedrich Schiller University Jena, Germany

²Michael Stifel Center Jena, Friedrich Schiller University Jena, Germany

Keywords: Machine Learning, Information Extraction, Provenance, Jupyter Notebook, Reproducibility.

Abstract: Machine learning (ML) pipelines are constructed to automate every step of ML tasks, transforming raw data into engineered features, which are then used for training models. Even though ML pipelines provide benefits in terms of flexibility, extensibility, and scalability, there are many challenges when it comes to their reproducibility and data dependencies. Therefore, it is crucial to track and manage metadata and provenance of ML pipelines, including code, model, and data. The provenance information can be used by data scientists in developing and deploying ML models. It improves understanding complex ML pipelines and facilitates analyzing, debugging, and reproducing ML experiments. In this paper, we discuss ML use cases, challenges, and design goals of an ML provenance management tool to automatically expose the metadata. We introduce MLProvLab, a JupyterLab extension, to automatically identify the relationships between data and models in ML scripts. The tool is designed to help data scientists and ML practitioners track, capture, compare, and visualize the provenance of machine learning notebooks.


1 INTRODUCTION


ML algorithms train models on sample data to allow predictions and thus support decision-making. A machine learning pipeline consists of several steps to train a model and is used to manage and automate ML processes. These steps are iterated several times to improve evaluation metrics (e.g., accuracy, precision) of the model and achieve better results. Consequently, there is a constant change in each phase of the ML pipeline, resulting in significant differences in the outcome. This makes ML pipelines more complex to reproduce and understand.


Provenance and metadata play vital roles in the reproducibility of results. The provenance of a data product is the description of the entities and the processes/steps together with the data and parameters that led to its creation (Herschel et al., 2017). Metadata is the data about data. Missing information about the development of proposed methods, data, and results can influence reproducibility. In ML, it is crucial to understand the data lineage to recognize why some predictions were made. It should be clear which data was explicitly used, how it got manipulated, and what

changes were made over time. To make scientific experiments reproducible, it is important to track information on the evolution of the code and its structure (*Definition Provenance*), the execution environment, including the system, external dependencies (*Deployment Provenance*), and the execution itself like variable values, outputs, run time information, etc. (*Execution Provenance*) (Pimentel et al., 2015). Definition, deployment, and execution provenance can also be used for enabling reproducibility of ML pipelines.

In this paper, we describe metadata and provenance management for end-to-end ML pipelines. We discuss which provenance information is required for the reproducibility of ML experiments. We present the design goals for our tool that support reproducibility and provenance management of ML models. In this regard, we introduce our proof of concept, called MLProvLab, which supports the design goals and provides a framework to capture, manage, compare, and visualize the provenance in notebook code environments, i.e., JupyterLab¹ (Kluyver et al., 2016). We discuss our evaluation plan and future work to support metadata and provenance management of end-to-end ML pipelines.

^a <https://orcid.org/0000-0002-0680-5753>

^b <https://orcid.org/0000-0002-7981-8504>

^c <https://orcid.org/0000-0002-2382-9722>

¹<https://jupyterlab.readthedocs.io>

2 BACKGROUND AND RELATED WORK

With the fast development of ML algorithms and the easy accessibility of ML frameworks and infrastructure, there is a growing need for provenance and model management from the ML community. There exists increasing attention on reproducibility not only in fields like biology, chemistry (Baker, 2016; Samuel and König-Ries, 2021), but also in ML and AI (Hutton, 2018; Raff, 2019). Schelter et al. (Schelter et al., 2018) present an overview of conceptual, data management, and engineering challenges in the ML model management. Automatically tracking and querying model metadata is one of the data management challenges with respect to the provenance management of ML. However, many existing ML frameworks have not been designed to automatically track provenance.

In recent years, several tools have been developed as metadata capturing systems (Vartak et al., 2016; Zaharia et al., 2018). Versioning tools like Git help in managing definition provenance. However, they do not capture information on ML model metadata. Tools like noWorkflow (Pimentel et al., 2015) provide tracking and capturing provenance of python scripts in general. On the other hand, other approaches are deeply tied to the data and the models used in machine learning itself (Ormenisan et al., 2020a; Ormenisan et al., 2020b; Baylor et al., 2017; Olorisade et al., 2017; Vartak et al., 2016; Zaharia et al., 2018; Namaki et al., 2020). ModelDB (Vartak et al., 2016) is one such system that provides a feature to manage ML models with metadata logging of metrics, artifacts, tags, and user information. Some approaches directly look into the file system and collect provenance data based on file changes (Ormenisan et al., 2020a). This can help understand how files are specifically used in model creation. Some systems track detailed provenance data by depending on the users to understand their complex schema and integrate their code with the corresponding API provided by the system (Schelter et al., 2017). In general, these provenance capturing systems require the user to actively configure their code, e.g., annotate hyperparameters, functions, and operations. But often, users omit to configure and annotate their code due to extra time and effort required. Therefore, tools that automatically extract and manage metadata have an advantage over systems that require human intervention.

Vamsa (Namaki et al., 2020), available as a command-line application, tracks provenance from Python scripts without requiring any changes to users' code. For this, the tool depends on an external knowl-

edge base containing APIs of various ML libraries that need to be added manually. However, this tool does not provide user interactivity. Project Jupyter (Kluyver et al., 2016) provides different tools like Jupyter notebooks and JupyterLab, which are widely used in developing ML pipelines. They are used by beginners, experts, and practitioners to write simple to complex ML scripts in their everyday work. However, these notebooks do not directly provide general provenance capturing features, let alone ML model management. ProvBook (Samuel and König-Ries, 2018) is a recent tool developed as an extension for Jupyter notebooks to capture, manage, query, compare, and visualize user history with interactivity. It is essential to provide provenance management without changing the code environment for the user. It is also important that such platforms provide metadata management to all their users irrespective of their skills and experience in ML. JupyterLab is a great basis for such projects as has been shown in other works (Kery et al., 2019). Hence, in this paper, we target the users of JupyterLab and allow automatic provenance extraction from ML notebooks and user interactivity.

3 PROVENANCE OF ML PIPELINES

In this section, we briefly describe the ML pipeline and explain the metadata and provenance information required for the reproducibility of ML scripts. Metadata and provenance management of ML pipelines is the problem of tracking and managing metadata and provenance of ML steps and models so that they can be reproduced, analyzed, compared, and shared afterward.

An ML pipeline, which is a multi-step process, automates the workflow to produce an ML model². Figure 1 shows the different stages and the provenance required for an end-to-end ML pipeline. The pipeline consists of the following stages: data discovery, collection, preprocessing, cleaning, feature engineering, model building, training, and evaluation, deployment, and monitoring. In a manual workflow, where no additional ML infrastructure is required, these steps are often performed in notebooks or scripts. The notebook/script is either executed locally or remotely to produce an ML model, which is the output of the pipeline. After the data discovery phase, raw data which is collected from different sources needs to be brought in the form ready for an ML task. For this

²<https://cloud.google.com/architecture/data-preprocessing-for-ml-with-tf-transform-pt1>

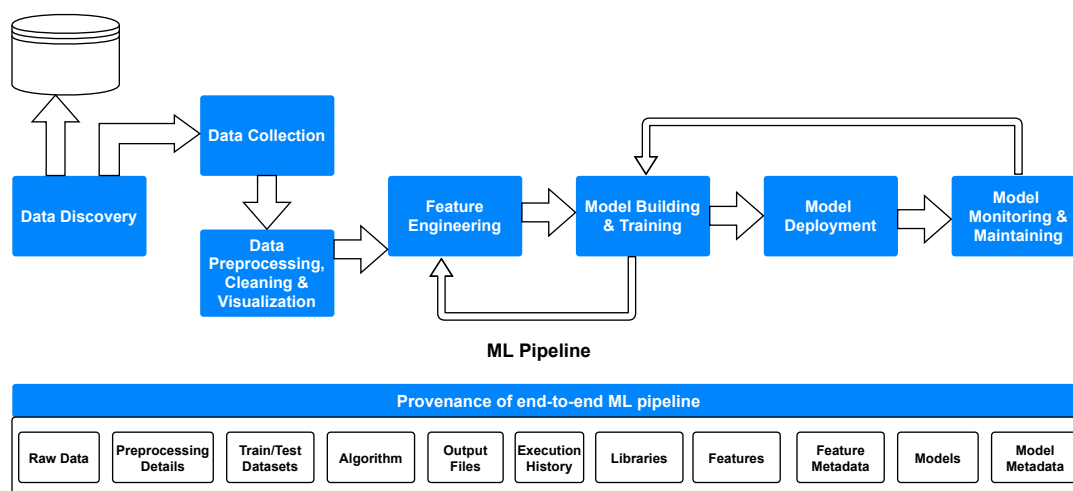


Figure 1: Provenance of end-to-end ML pipeline.

transformation, the raw data needs to be converted to processed data which involves data engineering operations. The processed data is then tuned to create engineered features for the ML model using feature engineering. The preprocessing stage contains several sub-steps, which are essential but often ignored by scientists for provenance documentation. Corrupted, invalid, or missing values need to be removed or corrected from the raw data in the data cleaning step. In the next step, the data points are selected and partitioned to create training, validation, and test datasets, using different techniques like random sampling, stratified partitioning, etc. Based on different ML problems, this phase involves further operations like tuning, extraction, selection, and construction of features using different methods and algorithms. After the data and feature engineering stages, the training, evaluation, and test sets are used to train the model. The trained model is then deployed. This is later then monitored and maintained.

In ML, the building and training of models is an iterative process. It requires several iterations to achieve results that satisfy acceptance criteria like accuracy, precision, etc. This workflow is ad-hoc, and there exist several challenges in managing models build over several iterations. Reproducibility is a time-consuming task, especially for ML pipelines, where model building and training can span hours or days. Hence, it is essential for data scientists to track and manage not only the results but also the parameter combinations used in the various stages of previous ML experiments. The paper (Olorisade et al., 2017) presents a set of factors that affect reproducibility in ML-based studies focusing on text mining. Another paper (Pineau et al., 2020) introduces a checklist required for reproducibility in the submission process

of an ML publication. Inspired by these works, we describe here a set of factors required for provenance management of ML applications developed in notebook code environments.

The provenance of the complete ML pipeline needs to be tracked to answer questions like *How was the model trained?*, *Which are the hyperparameters used?*, *Which features were used?*, *Where did the features come from?*, *Where did the bias come from?*, etc (Samuel et al., 2021). Raw data, preprocessing details, train/evaluation/test datasets, methods, algorithms, features, feature metadata, model, model metadata, execution history, random seeds, execution environment information, etc. are some of the important artifacts that need to be tracked for the reproducibility of an end-to-end ML pipeline (Fig. 1). The metadata, e.g., the location, version, size, and purpose of the datasets used, should also be tracked. This is helpful to identify any discrepancy that could occur in the result in the later experiments if there is a change in the datasets in the original location. The data transformation operations which convert raw data to engineered features are often overlooked during their documentation and publication. The provenance information in this step is crucial. Another important factor is to track how the dataset is partitioned to create training, validation, and test datasets. Algorithms, code, and the parameters used in the model building and training stage need to be captured. Randomization plays a crucial role in many ML algorithms, which can affect the end result. Therefore, it is crucial to set or use pseudo-random alternatives that allow deterministic behavior and thus produce same results and allow reproducibility. Execution environment like software and hardware information are other important provenance data. This includes information

on the programming language, kernel, versions, operating system, and machine type (CPU, GPU, cloud, etc.). The execution history, which explains what happened in each run of an ML pipeline, is another critical piece of information required by data scientists for the reproducibility of ML models.

4 DESIGN GOALS

We list here the design goals and the features for the proposed tool in JupyterLab for the metadata and provenance management of an end-to-end ML pipeline:

Support the provenance lifecycle. The provenance enabled life cycle management of ML experiments is the key factor required for reproducibility. Hence the tool should support the following provenance criteria:

- **Tracking:** The provenance information should be automatically extracted from the notebook and provided to users. This information includes the data, intermediate results, parameters, methods, algorithms, steps, execution history, and final results of the ML pipeline as mentioned in Section 3. In addition, the tool should also automatically identify the dependencies between variables, functions, etc., among different cells of a notebook.
- **Storage:** The tool should provide an efficient way to store the collected provenance.
- **Querying:** The collected provenance data should be made available to query. This would help users to answer questions like ‘Which dataset was used for building the ML model?’.
- **Compare:** The tool should provide users the feature to compare different versions of the execution of notebooks. This will help to compare results from the original one.
- **Visualization:** To support usability, users should be able to visualize the provenance in a way that they can understand how and why the result has been derived.

Support Reproducibility. The provenance information should help not only the user but also others to repeat and reproduce the results. With different versions of code, data, and execution history, we envision the tool to provide the ability to reproduce the notebook to run the ML pipeline for getting the original results.

Support Collaboration. We expect collaboration support among researchers by sharing the Jupyter

notebooks along with the provenance information of the ML pipeline.

Support Semantic Annotation and Interoperability. To aid interoperability, the tool should be able to support semantic annotation of ML pipelines using ontologies. We intend to describe the provenance information using the REPRODUCE-ME ontology (Samuel, 2019).

Support Exporting Provenance in Different Formats. According to the FAIR data principles, even if the data is deleted or removed for privacy concerns, the metadata should be made available (Wilkinson et al., 2016). Hence, we intend to provide support for exporting the provenance information. The provenance information can be exported in different formats, e.g., JSON, JSON-LD, RDF, so that the data is available for querying.

Ease of Use. The tool should be able to support different target groups, including beginners, experts, etc. Users should also have the possibility to interact with the tool.

Support Extensibility. We intend to design the tool in a way that new features can be easily added. The tool can be extended with additional functionalities to support each phase of the ML pipeline.

5 MLProvLab

We introduce our proof of concept for the provenance management of end-to-end ML pipelines in a notebook code environment. We present our tool, MLProvLab, a JupyterLab extension, to track, compare, and visualize the provenance of ML notebooks, as motivated by our design goals. The tool is available as an open source extension for Jupyter notebooks³.

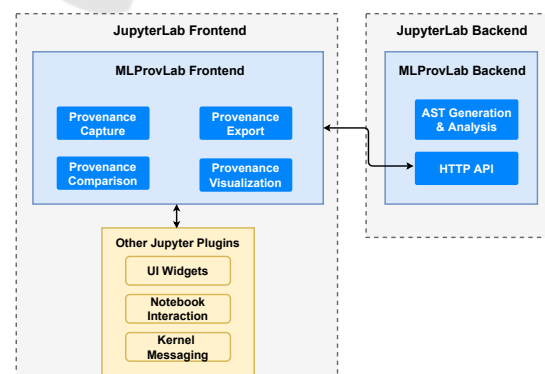


Figure 2: Architecture of MLProvLab.

Architecture. The MLProvLab tool is developed as an extension of JupyterLab so that it is available

³<https://github.com/fusion-jena/MLProvLab>

for data practitioners, researchers, and data scientists to support them in their daily work. JupyterLab is an open-source development environment for Jupyter Notebooks. Figure 2 shows the architecture of MLProvLab, which consists of a backend and a frontend plugin. The frontend mainly interacts with the core messaging plugin to get information from the kernel and the notebook panel. General visualization widgets are added in the frontend to display data to easily integrate it into the IDE layout. The MLProvLab tool is invoked to analyze the executed code with the help of an Abstract Syntax Tree (AST) and string pattern matching techniques to get data provenance. Fig-

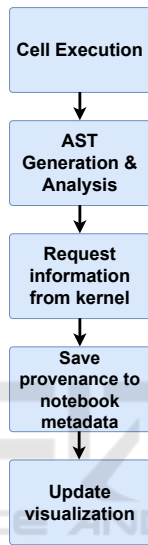


Figure 3: Workflow of MLProvLab.

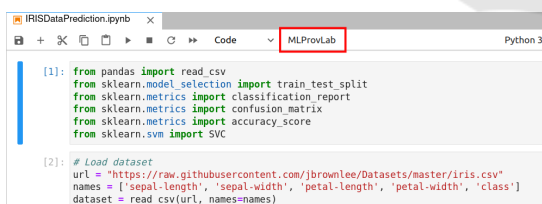


Figure 4: MLProvLab Toolbar button in JupyterLab.

ure 3 shows the workflow of MLProvLab. The tool defines event listeners that listen to different user actions like the execution, the addition and deletion of a cell. It generates an AST and analyses it, and then requests information from the kernel. The provenance information captured is saved to notebook metadata, and the visualization is updated.

Provenance Capture. MLProvLab collects and stores the provenance of a new user session triggered by the restart of a kernel as well as old user sessions (kernels). We call the lifetime of a kernel an *epoch*. Epochs are created for every new kernel and stored

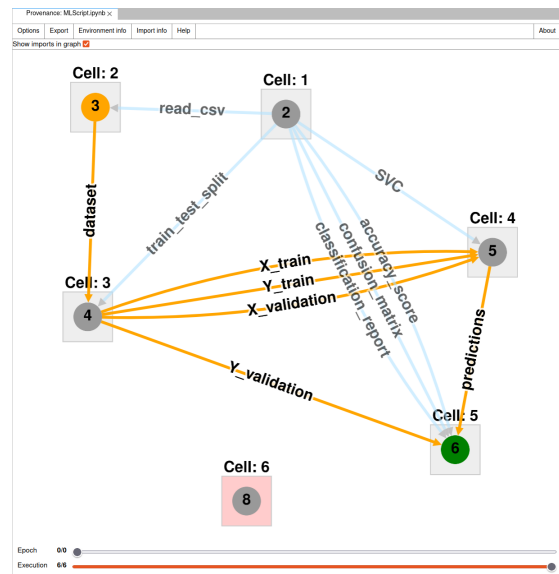


Figure 5: Main widget of MLProvLab.

in the provenance object in the notebook metadata. When the tool listens to the cell execution event, the code of the cell is sent back to the backend, which uses the Tornado web server⁴. We use AST for analyzing the code. Based on the information from the AST, we collect information on the definition and usages of variables, functions, and classes. We also track the import statements to extract information on the libraries and modules used along with their version information. In addition, the tool also tracks loops and conditions. We perform additional operations to find data sources for ML provenance management using string matching. Finally, we track every defined variable declared in the cell, the dependencies of variables that are not defined in the evaluated cell, used datasets and the corresponding variables, imported libraries, and modules, etc., as mentioned in Section 3. For the information collected using AST, we create a new object which contains the name of the called entity, a list of names with used entities, and other useful parameters such as position in code, etc. These are then combined and transferred back to the frontend, where it is inserted into a similar structured object. This also contains the execution count of the cell, cell id, outputs, and the executed code. Information request from the kernel about the defined variables in the cell is also added. This gives a snapshot of the state and its containing data. The newly created object is then stored in the corresponding order into the epoch where it was executed. This is then visualized to the user. In the first version of our tool, we include tracking, exporting, and visualization of

⁴<https://www.tornadoweb.org>

the provenance information of ML notebooks.

Provenance Visualization. MLProvLab uses a provenance graph to visualize the provenance of the notebook, including the execution order of cells and the data dependencies between cells. A new node is created in the graph for every new cell. New edges are created to connect the cell nodes. MLProvLab provides users the flexibility to choose the visualization based on the execution order of cells or the data dependencies between cells. Colors of the nodes and edges are updated accordingly based on the content of the cells, possible outputs, and data sources.

Figure 4 shows the MLProvLab extension in JupyterLab. The tool can be invoked using the ‘ML-ProvLab’ button in the notebook toolbar. By invoking the button, the main widget is opened containing the provenance graph. Figure 5 shows the provenance visualization graph of a sample ML notebook. The data sources and execution provenance are shown in the graph. Two sliders are provided at the bottom of the widget. The ‘Epoch’ slider provides the history of the execution of the Jupyter Notebook every time a new user session of the kernel is started. The ‘Execution’ slides provide the history of the execution of the Jupyter Notebook every time an event on the cell of the notebook is registered. The tool also shows the number of user sessions, executions, and execution time. Users are provided with a general menu with several options to customize the graph to get additional provenance information. The graph is built using Cytoscape.js (Franz et al., 2016). Cytoscape.js is well optimized and can display a large number of nodes and edges with little impact on performance. With its features, users can zoom in and out and get more information on each graph node.

For each cell in the notebook, a corresponding node is created in the graph. Detailed information on the latest execution of the cell is obtained based on the selected time frame on the bottom of the widget. Cells that contain data sources are displayed in orange. While the cells that contain output are colored green. Users can also change options in the menu bar to show the imported libraries and modules. It also shows in which cells the libraries and modules are used and provides information on those imported but not used in the notebook. Further provenance information can be visualized through a radial context menu. It can be opened by a right-click on a node or an edge. Clicking on a node, the user can select to visualize the definition provenance. This gives detailed information on the used datasets, functions, variables, etc. Users can also compare the definition provenance from previous runs as well. Figure 6 provides the information of this widget. The data displayed is the

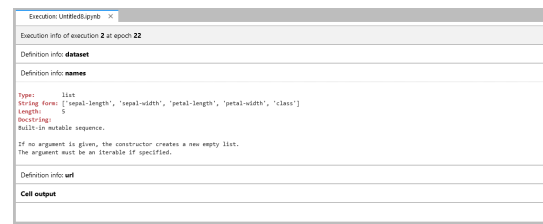


Figure 6: Definition and execution provenance widget.

plain text gathered from an information request to the kernel for each definition after a cell execution. Clicking on an edge gives users information on the specific variable. Edges that connect the cells with the dependencies are colored orange for data sources and blue for libraries and modules. This makes it easier for the user to track the whole flow of data from the input to the final output (Fig. 5). Figure 7 shows the execution environment information of the ML notebook. This includes information about the system, kernel, the used programming language, and its version for the currently selected epoch.



Figure 7: Execution environment information widget.

Provenance Comparison. Figure 8 shows the code difference widget for cells in a notebook. Users can explore the changes that were made to the code of a cell. With the slider on the bottom, users can select the previous ML experiments. This is visualized using the react-diff-view⁵ component.



Figure 8: Code difference widget.

Provenance Export. Users can export the provenance information of the ML notebook. Users can also clear the provenance history. However, users are provided with an alert to export the provenance before removing the provenance history from the notebook. The provenance information is currently available in

⁵<https://github.com/otakustay/react-diff-view>

JSON format. In the future, we plan to make this information available in other formats, including JSON-LD, RDF, etc., for semantic interoperability.

The MLProvLab tool will be released as an open-source extension for JupyterLab with an MIT license. Since it is a work-in-progress tool, we aim to implement all the features of the provenance management of end-to-end ML pipeline as discussed in Section 4 in our future work. We could use ML itself to analyze the work that is being tracked to give information about how the performance is or where problems could emerge. We plan to use logs and logging metrics for more information on gathering the provenance of ML models. We plan to do an extensive user evaluation to understand the user behavior and improve the tool. We also plan to do a performance-based evaluation with the publicly available notebooks in GitHub.

6 CONCLUSIONS

Jupyter notebooks are widely used by data scientists and ML practitioners to write simple to complex ML experiments. Our goal is to provide metadata and provenance management of the ML pipeline in notebook code environments. In this paper, we introduced the design goals and features required for the provenance management of the ML pipeline. Working towards this goal, we introduced MLProvLab, an extension in JupyterLab, to track, manage, compare, and visualize the provenance of ML scripts. Through MLProvLab, we efficiently and automatically track the provenance metadata, including datasets and modules used. We provide users the facility to compare different runs of ML experiments, thereby ensuring a way to help them make their decisions. The tool helps researchers and data scientists to collect more information on their experimentation and interact with them. This tool is designed so that the user need not to change their scripts or configure with additional annotations. In our future work, we aim to analyze metadata in more detail. We aim to track data sources by hooking into the file system or the underlying functions in the programming language itself. This will be integrated in a way that the user experience and performance are not compromised. We plan to use this provenance information to replay and rerun a notebook.

ACKNOWLEDGEMENTS

The authors thank the Carl Zeiss Foundation for the financial support of the project ‘A Virtual Werkstatt

for Digitization in the Sciences (K3)’ within the scope of the program line ‘Breakthroughs: Exploring Intelligent Systems for Digitization - explore the basics, use applications’ and University of Jena for the IMPULSE funding: IP-2020-10.

REFERENCES

- Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature News*, 533(7604):452.
- Baylor, D., Breck, E., Cheng, H., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., Koo, C. Y., Lew, L., Mewald, C., Modi, A. N., Polyzotis, N., Ramesh, S., Roy, S., Whang, S. E., Wicke, M., Wilkiewicz, J., Zhang, X., and Zinkevich, M. (2017). TFX: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1387–1395.
- Franz, M., Lopes, C. T., Huck, G., Dong, Y., Sumer, O., and Bader, G. D. (2016). Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2):309–311.
- Herschel, M., Diestelkämper, R., and Ben Lahmar, H. (2017). A survey on provenance: What for? what form? what from? *The VLDB Journal*, 26(6):881–906.
- Hutson, M. (2018). Artificial intelligence faces reproducibility crisis. *Science*, 359(6377):725–726.
- Kery, M. B., John, B. E., O’Flaherty, P., Horvath, A., and Myers, B. A. (2019). Towards Effective Foraging by Data Scientists to Find Past Analysis Choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, Glasgow Scotland Uk. ACM.
- Kluyver, T., Ragan-Kelley, B., et al. (2016). Jupyter notebooks—a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90.
- Namaki, M. H., Floratou, A., Psallidas, F., Krishnan, S., Agrawal, A., Wu, Y., Zhu, Y., and Weimer, M. (2020). Vamsa: Automated provenance tracking in data science scripts. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1542–1551.
- Olorisade, B. K., Breerton, P., and Andras, P. (2017). Reproducibility in machine learning-based studies: An example of text mining.
- Ormenisan, A. A., Ismail, M., Haridi, S., and Dowling, J. (2020a). Implicit provenance for machine learning artifacts. *Proceedings of MLSys*, 20.
- Ormenisan, A. A., Meister, M., Buso, F., Andersson, R., Haridi, S., and Dowling, J. (2020b). Time travel and provenance for machine learning pipelines. In Tala-gala, N. and Young, J., editors, *2020 USENIX Conference on Operational Machine Learning, OpML 2020, July 28 - August 7, 2020*. USENIX Association.

- Pimentel, J. a. F. N., Braganholo, V., Murta, L., and Freire, J. (2015). Collecting and analyzing provenance on interactive notebooks: When ipython meets no workflow. In *Proceedings of the 7th USENIX Conference on Theory and Practice of Provenance*, TaPP'15, page 10, USA. USENIX Association.
- Pineau, J., Vincent-Lamarre, P., Sinha, K., Larivière, V., Beygelzimer, A., d'Alché Buc, F., Fox, E., and Larochelle, H. (2020). Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *arXiv preprint arXiv:2003.12206*.
- Raff, E. (2019). A step toward quantifying independently reproducible machine learning research. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 5486–5496.
- Samuel, S. (2019). *A provenance-based semantic approach to support understandability, reproducibility, and reuse of scientific experiments*. PhD thesis, University of Jena, Germany.
- Samuel, S. and König-Ries, B. (2021). Understanding experiments and research practices for reproducibility: an exploratory study. *PeerJ*, 9:e11140.
- Samuel, S. and König-Ries, B. (2018). Provbook: Provenance-based semantic enrichment of interactive notebooks for reproducibility. In *International Semantic Web Conference (P&D/Industry/BlueSky)*.
- Samuel, S., Löffler, F., and König-Ries, B. (2021). Machine learning pipelines: Provenance, reproducibility and FAIR data principles. In Glavic, B., Braganholo, V., and Koop, D., editors, *Provenance and Annotation of Data and Processes - 8th and 9th International Provenance and Annotation Workshop, IPAW 2020 + IPAW 2021, Virtual Event, July 19-22, 2021, Proceedings*, volume 12839 of *Lecture Notes in Computer Science*, pages 226–230. Springer.
- Schelter, S., Biessmann, F., Januschowski, T., Salinas, D., Seufert, S., and Szarvas, G. (2018). On challenges in machine learning model management. *IEEE Data Eng. Bull.*, 41:5–15.
- Schelter, S., Boese, J.-H., Kirschnick, J., Klein, T., and Seufert, S. (2017). Automatically tracking metadata and provenance of machine learning experiments. In *Machine Learning Systems Workshop at NIPS*, pages 27–29.
- Vartak, M., Subramanyam, H., Lee, W.-E., Viswanathan, S., Husnoo, S., Madden, S., and Zaharia, M. (2016). Modeldb: a system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pages 1–3.
- Wilkinson, M. D. et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data*, 3.
- Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., and Zumar, C. (2018). Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45.