MLProvCodeGen: A Tool for Provenance Data Input and Capture of Customizable Machine Learning Scripts

Tarek Al Mustafa [1], Birgitta König-Ries [2], Sheeba Samuel [3]

Abstract: Over the last decade or so Machine learning (ML) has dramatically changed the application of and research in computer science. ML approaches, mostly based on deep neural networks, can be found in many sectors, from automating tasks to decision making and from scientific workflow management to database systems. Large amounts of data, economical data storage, increased computation power, and improved algorithms have resulted in the rapid growth of ML. ML allows to solve complex problems with intricate solutions. This power comes at a price, however. Assuring transparency and reproducibility for advanced ML systems all the way from raw data to deployment, becomes increasingly complicated. In this paper, we describe an approach that flips the problem: Why not let users specify a variety of parameters that together provide complete provenance information via a user interface and automatically generate executable and reproducible ML code from this information? We introduce *MLProvCodeGen* (Machine Learning Provenance Code Generator), a *JupyterLab* extension to generate custom code for ML experiments using user-defined metadata. Users can automatically generate ML workflows with different data settings, model parameters, methods, and training parameters and reproduce results in *Jupyter Notebooks*. We evaluate our proof of concept implementation with two ML applications, image classification and multiclass classification, and conducted a user evaluation to strengthen our conclusions.

Keywords: Provenance Management; Code Generation; Machine Learning; JupyterLab; Jupyter Notebooks; Reproducibility

1 Introduction

Over the last decade, Machine Learning (ML), in particular approaches based on deep neural networks, has increasingly gained importance and is the by far domineering data science approach today. It can be seen in many sectors and can be used to solve various problems, but it also offers opportunities to benefit the scientific community at large by supporting scientific workflows [De19] and database systems [Ma20; Va17].

With this increased importance come new challenges in data engineering and management: Machine learning pipelines are used to streamline the process of producing ML models. ML workflows include steps to get results for given problems from raw data. These steps are

[1] Friedrich Schiller University Jena, Jena, Thuringia, Germany tarek.almustafa@uni-jena.de
[2] Friedrich Schiller University Jena, Michael Stifel Center Jena, Jena, Thuringia, Germany birgitta.koenig-ries@uni-jena.de
[3] Friedrich Schiller University Jena, Michael Stifel Center Jena, Jena, Thuringia, Germany sheeba.samuel@uni-jena.de

gathering, cleaning, and preprocessing data, feature engineering, model creation, training, evaluation, and deployment. Even though these steps are common for every ML workflow, the specifics of the implementation, metadata of the entire experiment, and provenance information differ for each ML model. This provenance information is critical as it documents the settings that determine the end results of ML experiments, making it a vital component for experiment reproducibility. If information about the data, methods, environment, or results of a ML experiment is missing, reproducibility can be inhibited.

In many areas of scientific study, but specifically in machine learning and artificial intelligence, reproducibility has become an increasingly important issue [Ba16; Hu18; SK21]. Building a complete and reproducible, ML pipeline from raw data to deployment is challenging, especially for new ML practitioners. Automatic code generation based on templates can reduce the complexities of creating ML models. With this method users can generate code for ML pipelines with different pre-defined parameters and variables. This approach also supports reproducibility, as template-based code generation can also include automatic provenance data capture at the same time.

In this paper, we aim to identify the key provenance information required to document an entire ML workflow from raw data to deployment. We use this information to build a system that can generate custom code for different ML applications. By adding provenance capture to the generated scripts, we support the reproducibility of results at any step of the process.

We introduce *MLProvCodeGen*, a custom code generation tool for ML experiments based on the identified key provenance information. Users can input a variety of data settings, model parameters, methods, training parameters, etc., and generate ML code using a template-based architecture. *MLProvCodeGen* is available as a *JupyterLab*[4] extension. *JupyterLab* is a web-based user interface that allows users to perform interactive and explorative computing. It also lets users add their own templates for other ML problems.

The main contributions of this work are: (1) define the minimum requirements to reproduce chosen ML workflows. (2) use these minimum requirements as a data model to build a template based system to automatically generate ML code in Jupyter notebooks[5] with multiple, user-chosen parameters. (3) automatically capture and display provenance data from the generated notebooks to allow one-to-one reproductions by (4) inputting captured data into our system.

Outline. In Sect. 2, we discuss background and related work. In Sect. 3, we describe *MLProvCodeGen*'s design goals, implementation, and data model in detail. Sect. 4 presents the user evaluation we conducted and Sect. 5 discusses our conclusions and future work.

[4] `https://jupyterlab.readthedocs.io`

[5] `https://jupyter-notebook.readthedocs.io/en/stable/`

2 Background and Related Work

In the last decades, many approaches have been proposed to improve and support reproducible computational research. In this section, we compile a summary of different branches of this research.

Provenance Data and Reproducibility. Provenance plays a key role in reproducibility [Mi16]. Using the same data and code to obtain the same results as mentioned in a given publication, is an important step to verify the reliability of research findings. According to this, the Neural Information Processing Systems (NeurIPS) conference introduced a *Machine Learning Reproducibility Checklist* in 2019 as part of the paper submission process [Pi21]. This checklist presents a list of information required for models and algorithms, theoretical claims, and results to be included in the submission process. Olorisade et al. [OBA17] identify key aspects of text mining experiments needed to facilitate reproduction, based on their attempt to reproduce six published studies. A different provenance checklist was a result of MIT's Human Data Interaction Project, containing model provenance specifications for time-based predictive ML problems[6].
Provenance data can be divided into two sub-types, prospective and retrospective provenance. The prospective provenance of a computational task describes its specification and steps that must be followed to generate a data product [Fr08]. Retrospective provenance captures what actually happened during the execution of a computational task. For the reproducibility of ML experiments, it is important that both provenance data types are captured and documented [De15; HDB17].
In our previous work, we have investigated factors beyond the availability of source code and datasets that influence reproducibility of ML experiments based on a survey conducted among ML practitioners [SLK20].

Scientific Workflow Management Systems. Scientific workflow management systems have been used in a variety of disciplines to support scientists in conducting experiments. These systems are especially useful for experiments that follow clearly defined workflows. These experiments include ML experiments, as their workflow can be structured through ML pipelines. Notable scientific workflow management systems include Galaxy [GNT10], Taverna [Oi04; Wo13], Kepler [Lu06], and VisTrails [Ca06; Fr18], a system to support reproducibility by scientific vizualization of dataflows and data transformations through provenance capture. The core function of many of these systems is to break down complex workflows into smaller, more manageable parts, just as we aim to do with our system by separating ML pipelines into singular steps.
Provenance Data Models. Provenance data models are used to specify the format in which metadata is presented and which specific data points can be represented for a given application. Each application of provenance metadata relies on a data model. Therefore, scientists might define their own provenance data model catered to the application. However, there has been an effort to unite the most common data points into standardized specifications

[6] `https://github.com/HDI-Project/model-provenance-json`

that scientists can use. Notable examples include the first workshop on the *Open Provenance Model* in 2008 [Mo08] resulting in what would later be released as the *Open Provenance Model core Specifications* in 2011 [Mo11]; the *Report from the Dagstuhl Semiar of March 2012 on the Principles of Provenance* [Ch12]; and the efforts of the World Wide Web Consortium (W3C), resulting in the *W3C PROV family of specifications for modelling provenance metadata* [MBC13]. This family of specifications includes, but is not limited to, *The PROV Data Model (PROV-DM)* [Be13], the practical examples of *PROV-DM* and the specifications for serialization of *PROV* instances in *The Provenance Notation (PROV-N)* [Mo13], and *The Provenance Ontology (PROV-O)* [Le13], an encoding of *PROV-DM* into *OWL2 Web Ontology Language*.
Provenance Ontologies. Provenance ontologies are used to present provenance data in a machine-readable way. *PROV-O* has been mentioned above as an encoding of the *PROV-DM* specification. *ML-Schema* is a "top-level ontology that provides a set of classes, properties, and restrictions for representing and interchanging information on machine learning algorithms, datasets, and experiments"[Pu18, p. 1] and presents useful terms and vocabulary for the ML domain. The *REPRODUCE-ME Ontology* [SK17; SK18a] extends *PROV-O* and later includes *provenance-plan (P-PLAN)*[7] vocabulary to enhance the reproducibility of microscopy experiments by allowing semantic queries on microscopic data. The core function of the *REPRODUCE-ME Ontology* is to describe all computational and non-computational steps and data of scientific experiments.

Provenance Capture Systems and applications of Provenance Data Models. In recent years there have been a number of applications of these standardized specifications and ontologies which may adopt or adjust existing data models. Other significant works include *PROV-ML* defined in [So19], which uses *W3C PROV* and *ML-Schema* to specify a provenance data model for complex exercises in the computational science and engineering domain; the *MEX Vocabulary* [Es15] to explain provenance information in ML, based on *PROV-O* and following Linked Data best practices; and multiple systems that aim to capture provenance data automatically from either ML scripts [Na20; Sc17], model outputs [Ma17], computational notebooks [SK18b; SK20], specific workflow steps like data cleaning [PML20], or whole systems [Sc18]. We have reviewed data models proposed by the applications above and derived the data model of our system from them.

Code Generation and Templates. Automatic code generation can increase productivity and consistency in ML scripts. Code generation tools can assist the development of automatic programming tools to improve programming productivity [LCB20]. Once the code for an ML application can be written with a system using automatic code generation, it can be reused multiple times with different parameters and users have a guarantee that code resulting from this method will always be identical. However, implementing a system that supports automatic code generation severely impacts the complexity of the whole exercise and it forces the system to have a frontend that provides the system with the information to be used in the workflow. A recently developed tool called *TrainGenerator* [Ri21] provides and

[7] `http://vocab.linkeddata.es/p-plan/version/13032014/`

generates custom template code for ML. It offers multiple options for preprocessing, model setup, training, and visualization. The workflow of *TrainGenerator*, utilizing a frontend user interface to capture information used in the backend to select templates for code generation, heavily inspired our work. Another recent work exploring automatic code generation is a project called *GitHub Copilot*[8] that takes user prompts as input and suggests code based on these prompts.

3 MLProvCodeGen

In this section, we introduce *MLProvCodeGen*, a *JupyterLab* extension which allows users to specify a variety of provenance related parameters. From these, executable and reproducible ML code is automatically generated. As a proof-of-concept, we support two different ML problems, namely Image Classification and Multiclass Classification, each with their set of customizable parameters. The generated scripts are in *Jupyter Notebook* format. *MLProvCodeGen* supports the end-to-end reproducibility of ML pipelines by integrating provenance data capture into the generated notebooks. The provenance data captured by *MLProvCodeGen* can be exported to *JSON* format, or shown in a provenance graph. Exported provenance data from *MLProvCodeGen* can be input into the system to deliver an identical reproduction of the ML experiment.

3.1 Design Goals

The design goals and features of *MLProvCodeGen* are:

Automatic Code Generation. Users should be able to successfully generate code for the selected ML problem. The generated code should be able to run error-free.
Customizability. Users should be able to select between different ML exercises and customize the generated scripts through a wide variety of parameters.
Provenance Data Model. Our systems provenance data model should comply with existing provenance data standards and specifications.
Provenance Data Capture. The system should be able to capture provenance data from generated scripts. The provenance data captured should be sufficient to reproduce the experiment.
Reproducibility. ML experiments of which provenance data was captured should be reproducible with our system by using the captured provenance data as its input.
Addition of new templates. Users should be able to extend the system by adding and customizing templates for other ML problems. The tool should provide an easy workflow for the addition of new templates.
Comparison of results between different models. Users should be able to compare results

[8] https://github.com/features/copilot

between different models for the same ML problem. Hence, a provenance management system to track the results from different executions should be integrated into the system.
Organization of code. Poor programming practices can hinder the reproducibility and benefits of notebook programming. Hence, the tool should organize the generated code logically, break into reusable modules, and define the title, purpose, installations, import statements, and variables at the top of the notebook, respectively.

3.2 System Architecture

MLProvCodeGen is developed as an extension of *JupyterLab* to make it accessible for data practitioners, researchers, and data scientists. Fig. 1 shows the architecture of *MLProvCode-Gen*. It consists of a frontend plugin to capture information, and a backend plugin to process

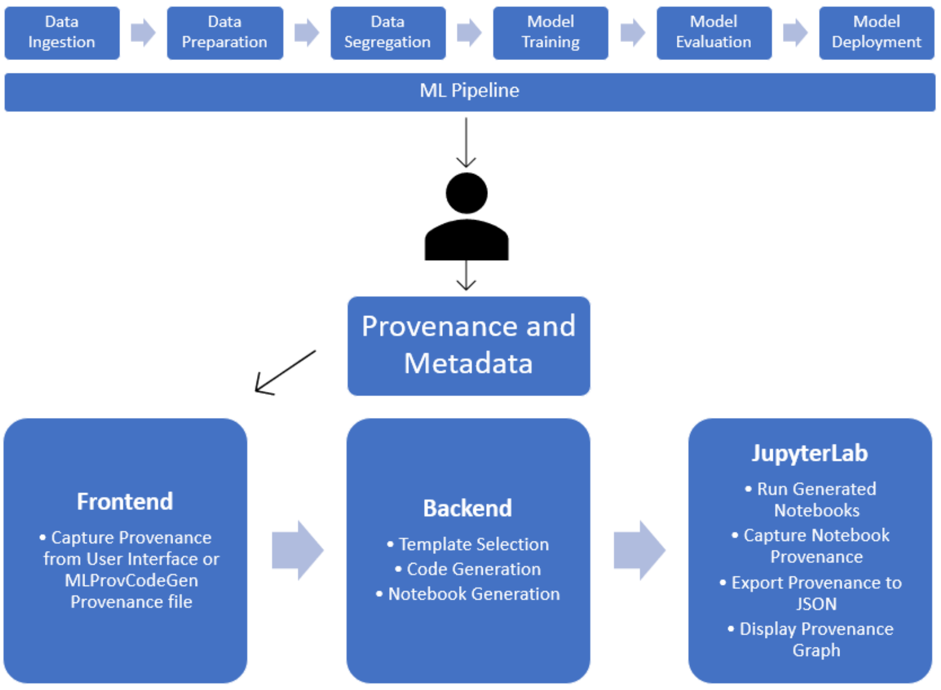
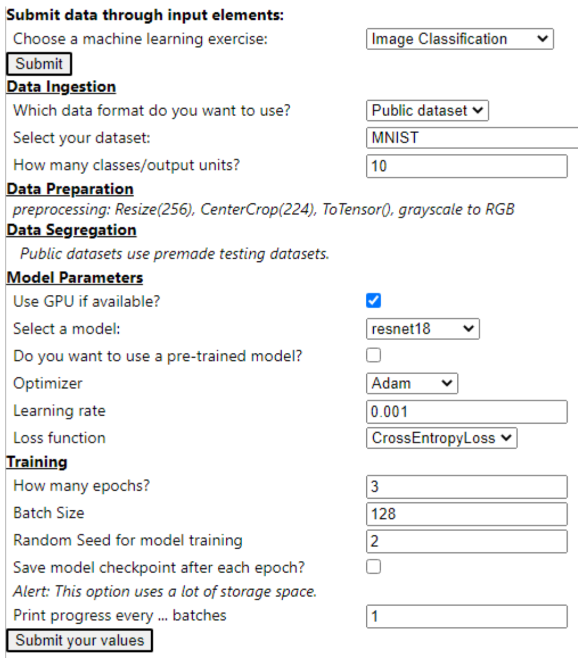


Fig. 1: System Architecture of MLProvCodeGen

that information and generate notebooks from it. In detail, *MLProvCodeGen*'s workflow is as follows:

Frontend. The frontend plugin of *MLProvCodeGen* provides a user interface as shown in Fig. 2. The variety of parameters in the user interface for each implemented ML application is listed below in Sect. 4.1. Users can open the extension by clicking the *MLProvCodeGen*



Submit data through input elements:

Choose a machine learning exercise:	Image Classification

Submit

Data Ingestion

Which data format do you want to use?	Public dataset
Select your dataset:	MNIST
How many classes/output units?	10

Data Preparation

preprocessing: Resize(256), CenterCrop(224), ToTensor(), grayscale to RGB

Data Segregation

Public datasets use premade testing datasets.

Model Parameters

Use GPU if available?	✓
Select a model:	resnet18
Do you want to use a pre-trained model?	☐
Optimizer	Adam
Learning rate	0.001
Loss function	CrossEntropyLoss

Training

How many epochs?	3
Batch Size	128
Random Seed for model training	2
Save model checkpoint after each epoch? *Alert: This option uses a lot of storage space.*	☐
Print progress every ... batches	1

Submit your values

Fig. 2: Image Classification Input Elements in the User Interface

button in the *other* section of *JupyterLab*'s home interface. They can also open the extension by entering a custom command in the *JupyterLab* command pallete. At the bottom of the interface, users can submit their inputs so that they can be packaged and subsequently sent to the backend plugin. The user interface also allows users to input a provenance file from an experiment generated by *MLProvCodeGen* in the past in order to reproduce it.

Backend. Given user inputs from the frontend, the backend selects the appropriate function to execute. While the workflow of the use cases is similar, each use case has its own function in the backend. Provenance information is extracted from the data package so that it can be used for templating. *MLProvCodeGen* generates notebooks using custom code templates. *Jinja*[9] is used as the templating language as it is a fast, expressive, and extensible templating engine for the *Python* programming language. It supports modular templates, variables, control flow, etc. and creates documents based on the provided data. The backend plugin first generates a new and empty notebook file. The empty notebook is filled with markdown and code cells, each having their own templates and variables associated with it. When code generation is complete, the notebook is saved with the name of the ML problem that the code solves. Upon successful generation of the notebook, *MLProvCodeGen* returns a success message to the user and provides them with a button to open it.

[9] `https://jinja.palletsprojects.com/en/3.1.x/`

Notebooks. The notebooks are structured as follows: At the top is a markdown cell containing information about the ML exercise itself. The following code cells contain the installation command of the requirements and packages needed to run the notebook. Import statements are added directly after and provenance data capture is initialized. The remaining cells follow the structure of an ML pipeline. Each notebook has a cell for data ingestion, data preparation, data segregation, the model, training, and evaluation. At the bottom of the notebooks are cells to generate a provenance graph, generate a provenance data file in JSON format, and view the file and graph.

experiment_info	creation_date, file_size, modification_date, task_type, title
hardware_info	CPU, GPUs, Operating_System, RAM
packages	All Python packages used in the notebook + the package version used
notebook	prov:type, creation_date, file_format, name, kernel, programming_language, programming_language_version
data_ingestion	start_time, end_time, execution_time, data_format, dataset_id, dataset_classes, feature_dimensions, dataset_description, root_location, training_samples, testing_samples, validation_samples
data_preparation	start_time, end_time, execution_time, number_of_operations, operations
data_segregation	start_time, end_time, execution_time, training_split, testing_split, validation_split
model_parameters	start_time, end_time, execution_time, gpu_enable, pretrained, save_checkpoint, model_name, model_description, activation_function, output_neurons, loss_function, optimizer, optimizer_learning_rate
model_training	start_time, end_time, execution_time, random_seed, resulting_model_seed, batch_size, epochs, print_progress
model_evaluation	start_time, end_time, execution_time, evaluation_metrics(accuracy, loss, AUC, Confusion Matrix, F1, MAE, MSE)

Tab. 1: Provenance Data Model of MLProvCodeGen

Provenance Data Capture. Provenance data of the notebooks is captured using the *prov*[10] Python package. This allows us to specify entities, agents, and activities according to *PROV-DM* specifications and build p-plans and collections adjacent to *PROV-O*. Provenance information is captured about the experiment, notebook file, programming environment, packages, and system specifications. If a specific function was used to capture that information, *MLProvCodeGen* generates an activity describing it. Each code cell is an entity, has an activity that describes the execution of that cell, and a second entity that describes the data generated by the execution of that cell. Cell entities are ordered by specifying how a given cell was influenced by the ones executed before it. All provenance information captured for notebooks generated by *MLProvCodeGen* is listed in Tab. 1. At the end of the notebook, the captured provenance data is used to generate a graph visualisation showing all data points, as well as saving the data in a JSON file. The image and data files are named after the ML problem and time at which they were generated. However, capturing provenance data using the prov package is not automatic and has to be written into the notebook's code.

[10] `https://pypi.org/project/prov/`

This means that all changes made to the code after the notebook's generation have to be added to the provenance capture manually, which presents a possible downside.

Provenance Data Input. Any provenance data file generated by *MLProvCodeGen* can be uploaded to the system in the user interface to generate an identical reproduction of the code described by the provenance data. Uploaded files are processed in the backend in the exact same way as data input by users through the input elements in the user interface.

4 Evaluation

We evaluated *MLProvCodeGen* in two parts. At first, we checked the feasibility of our solution by implementing two proof of concept examples, Image Classification and Multiclass Classification. Then we conducted a user evaluation using an online survey. Each of these two parts will be described in their own subsection.

4.1 Implementation Examples

Image Classification. Image Classification (IC) is one of the basic pattern recognition problems. Taking image files as input, a model trained for image classification will split the set of input images into a given number of classes. We use the *PyTorch* library [Pa19] for our implementation. Currently, an IC notebook can be generated for three public datasets, *MNIST*[11] [De12], *FashionMNIST*[12] [XRV17], and *CIFAR10*[13]. There are a variety of options for model parameters: three different models, *resnet18* [He16], *shufflenetv2* [Ma18], and *vgg16* [SZ14]; six different optimizers, *Adam* and *Adamax* [KB14], *Adadelta* [Ze12], *Adagrad* [DHS11], *RMSProp* [TH+12], and *SGD* [Su13]; as well as two loss functions, *CrossEntropyLoss*[14] and *NLLLoss*[15]. All other input elements are either checkboxes or number inputs that can be changed by users at their discretion. The notebooks that are generated from user inputs automatically, are able to be run error-free for all parameters. Provenance data is captured and shown in a provenance graph, as well as saved in *JSON* format.

Multiclass Classification. Multiclass Classification (MC) describes classification exercises with more than two target classes. Taking tabular data as input, a model trained for multiclass classification aims to predict the class of an input unit. *MLProvCodeGen* uses a configurable neural network and supervised learning to solve this exercise using both the *Scikit-learn* library [Pe11], and *PyTorch*. A MC notebook can be generated for four public datasets, *Iris*

[11] `http://yann.lecun.com/exdb/mnist/`

[12] `https://github.com/zalandoresearch/fashion-mnist`

[13] `https://www.cs.toronto.edu/~kriz/cifar.html`

[14] `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`

[15] `https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html`

flowers [16], *Spiral patterns*[17], *Aggregation*[17], and *R15*[17]. Users can also select an option to insert their own tabular data in *csv* format. Data Preparation for *MLProvCodeGen* currently supports a scaler[18]. Data Segregation allows users to input a random seed, and to define the size of training and testing datasets. The options for model parameters include: Three activation functions, *Softmax*[19], *Sigmoid*[20], *Tanh*[21]; a number input field to define the number of neurons used in the middle layer; six optimizers, *Adam*, *Adadelta*, *Adagrad*, *Adamax*, *RMSprop*, and *SGD*; a configurable learning rate; and three loss functions, *CrossEntropyLoss*, *NLLLoss*, and *MultiMarginLoss*[22]. Users can also change the number of training epochs. The notebooks that are generated from user inputs automatically, are able to be run error-free for all parameters. Provenance data is captured and shown in a provenance graph, as well as saved in *JSON* format.

4.2 User Evaluation

We conducted a user evaluation to measure *MLProvCodeGen*'s user experience by combining an online survey via *LimeSurvey*[23] and a virtual installation of our program via *Binder*[24]. Out of a total of 26 entrants, 12 entrants successfully completed the survey and another 2 entrants got to the halfway point. The remaining entrants left the survey before starting to work on the task. All questions and completed answers are available online[25].

The goal of the user evaluation was to test the appropriateness and general usability of *MLProvCodeGen* for users from the computer science domain who may or may not be familiar with machine learning experiments, data provenance, and reproducibility. We asked users to self assess their level of proficiency with these terms, to complete hands-on user tasks, and consequently rate their experience using a variety of metrics. Of the 12 participants, eight answered the question regarding their professional background. All eight of them had a background in computer science or a related field. Prior knowledge about both machine learning and reproducibility was very mixed with all values from "poor" to "excellent" selected (see Fig. 3). The distribution is skewed towards lower levels of expertise and experience with respect to machine learning, while the level of knowledge on reproducibility was quite evenly distributed and higher levels of knowledge were mentioned more frequently for FAIR data and scientific data management. We believe that this might reflect a higher

[16] `https://archive.ics.uci.edu/ml/datasets/iris`
[17] `http://cs.joensuu.fi/sipu/datasets/`
[18] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`
[19] `https://pytorch.org/docs/stable/generated/torch.nn.Softmax`
[20] `https://pytorch.org/docs/stable/generated/torch.nn.Sigmoid`
[21] `https://pytorch.org/docs/stable/generated/torch.nn.Tanh`
[22] `https://pytorch.org/docs/stable/generated/torch.nn.MultiMarginLoss.html?highlight=multimarginloss`
[23] `https://www.limesurvey.org/`
[24] `https://mybinder.org/`, available at `https://mybinder.org/v2/gh/fusion-jena/MLProvCodeGen/main?urlpath=lab`
[25] `https://github.com/fusion-jena/MLProvCodeGen/tree/main/EvaluationResults`

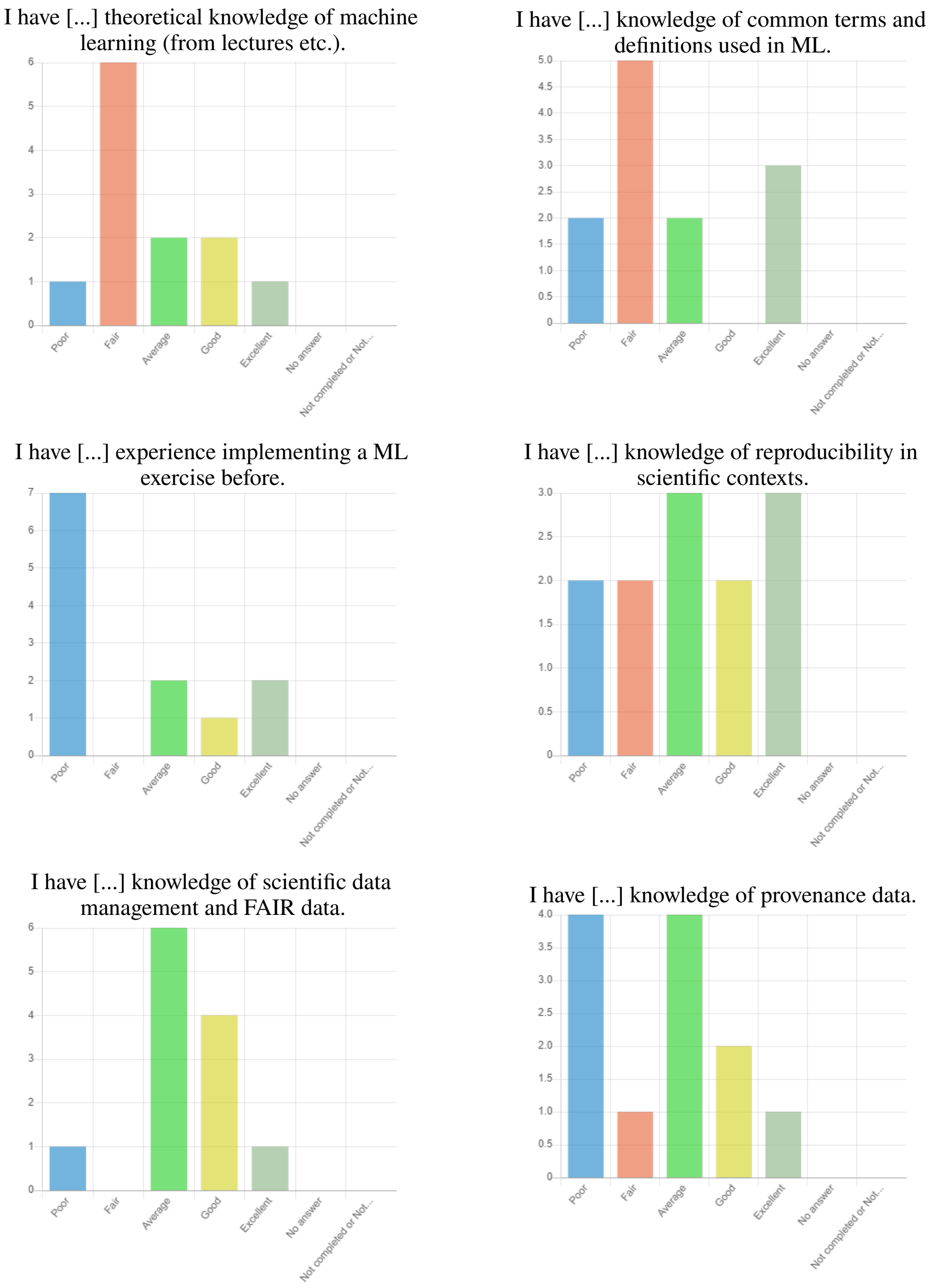


Fig. 3: Self assessment questions from the user survey

willingness of people with an awareness for the need of provenance management to take part in the survey compared to others.

In the first part of the survey, the overall system, important terms and the ML problems were explained to the participants. 11 out of 12 participants agreed or strongly agreed that they understood the terms used in the system to represent these. After this introduction, participants were required to run a notebook (including choosing a dataset and setting parameters (similar to Fig. 2). Participants were asked afterwards whether they understood the purpose of the code cells and of the code within each cell. A vast majority of users confirmed this; the result was a bit weaker for the second question but still positive.

We then asked participants to rate how helpful different visual elements were to understanding the notebook. Fig. 4, Fig. 5 and Fig. 6, show examples for some of them, namely the visualisations of the data, the training progress and the validation accuracy and loss.

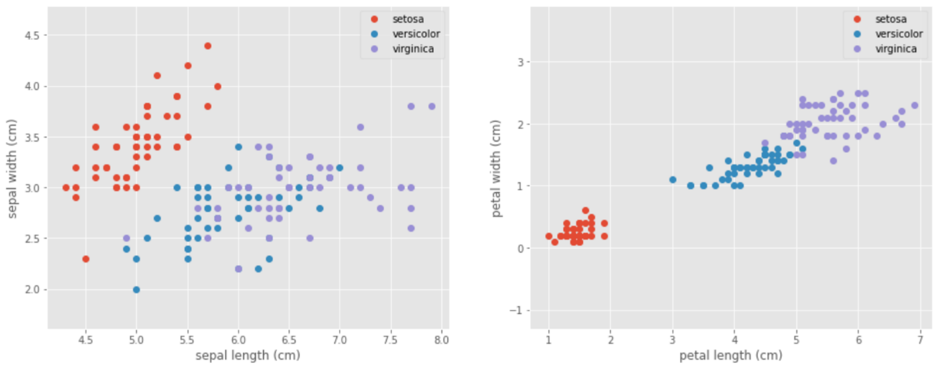
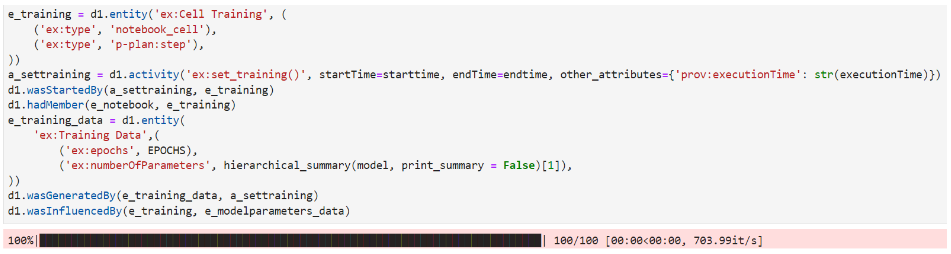


Fig. 4: Visualisation of a dataset in MLProvCodeGen



```
e_training = d1.entity('ex:Cell Training', (
    ('ex:type', 'notebook_cell'),
    ('ex:type', 'p-plan:step'),
))
a_settraining = d1.activity('ex:set_training()', startTime=starttime, endTime=endtime, other_attributes={'prov:executionTime': str(executionTime)})
d1.wasStartedBy(a_settraining, e_training)
d1.hadMember(e_notebook, e_training)
e_training_data = d1.entity(
    'ex:Training Data',(
        ('ex:epochs', EPOCHS),
        ('ex:numberOfParameters', hierarchical_summary(model, print_summary = False)[1]),
))
d1.wasGeneratedBy(e_training_data, a_settraining)
d1.wasInfluencedBy(e_training, e_modelparameters_data)
```

```
100%|████████████████████████████████████████| 100/100 [00:00<00:00, 703.99it/s]
```

Fig. 5: Visualisation of training progress in MLProvCodeGen

Fig. 7 shows the results to this set of questions confirming the positive role of all of these elements.

In the final part of the survey, we aimed to understand the usefulness and usability of the visualisation of the provenance graph (see Fig. 8 for a small excerpt. A full graph can be found at `https://github.com/fusion-jena/MLProvCodeGen/blob/main/`

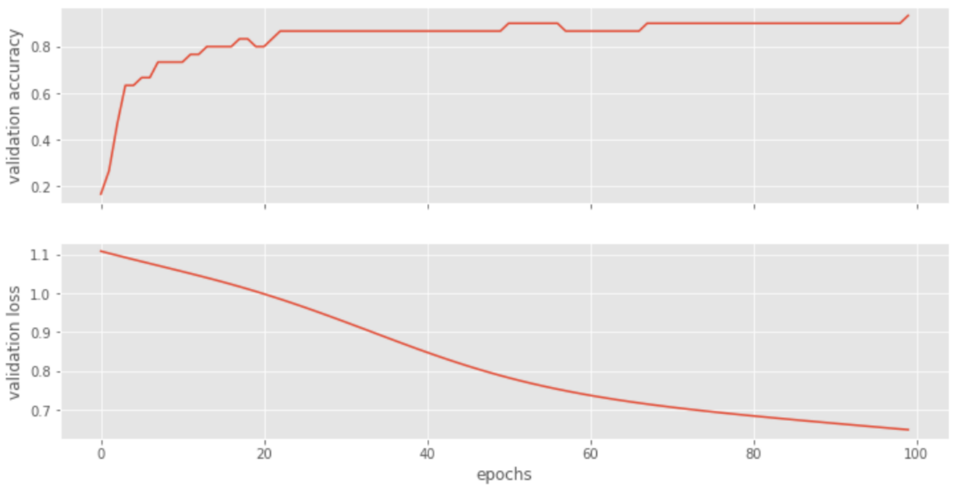


Fig. 6: Visualisation of validation accuracy and loss in MLProvCodeGen

`GeneratedProvenanceData/Provenance_MulticlassClassification_2022-05-19--11-09-37.png`).

Here, participants were asked to find specific values (the accuracy and the function that determines the creation date). Participants were then asked to judge the difficulty of these tasks and the provenance graph in general. These got pretty mixed ratings (see Fig. 9 also reflected in the free text comments users were able to provide (e.g., "Too much clutter, especially too many purely technical details in provenance graph; perhaps different views would be good, e.g. a data analyst vs a software engineering view –> less important information could be greyed out or hidden"). Despite these mixed ratings, the first task was successfully completed by 11 out of 12 participants, the second task was solved completely correct by 4 participants, 4 identified at least the correct region of the graph, 4 failed to complete the task.

Finally, we asked users about their overall experience with MLProvCodeGen. 11 agreed they had a good experience, 1 strongly agreed to this statement.

The key conclusions of the online survey are as follows: (1) The explanations and instructions given are adequate in order to use *MLProvCodeGen* without any outside help. (2) The user interface of *MLProvCodeGen* is intuitive and easy to use. (3) The generated notebooks have comprehensible structure and, depending on the users expertise, the code is coherent and understandable. (4) While the provenance graph displays the provenance data as intended, it is hard to navigate within the graph, to find singular data points, and to gain an overall understanding. The visualization of captured provenance data leaves the most room for improvement.

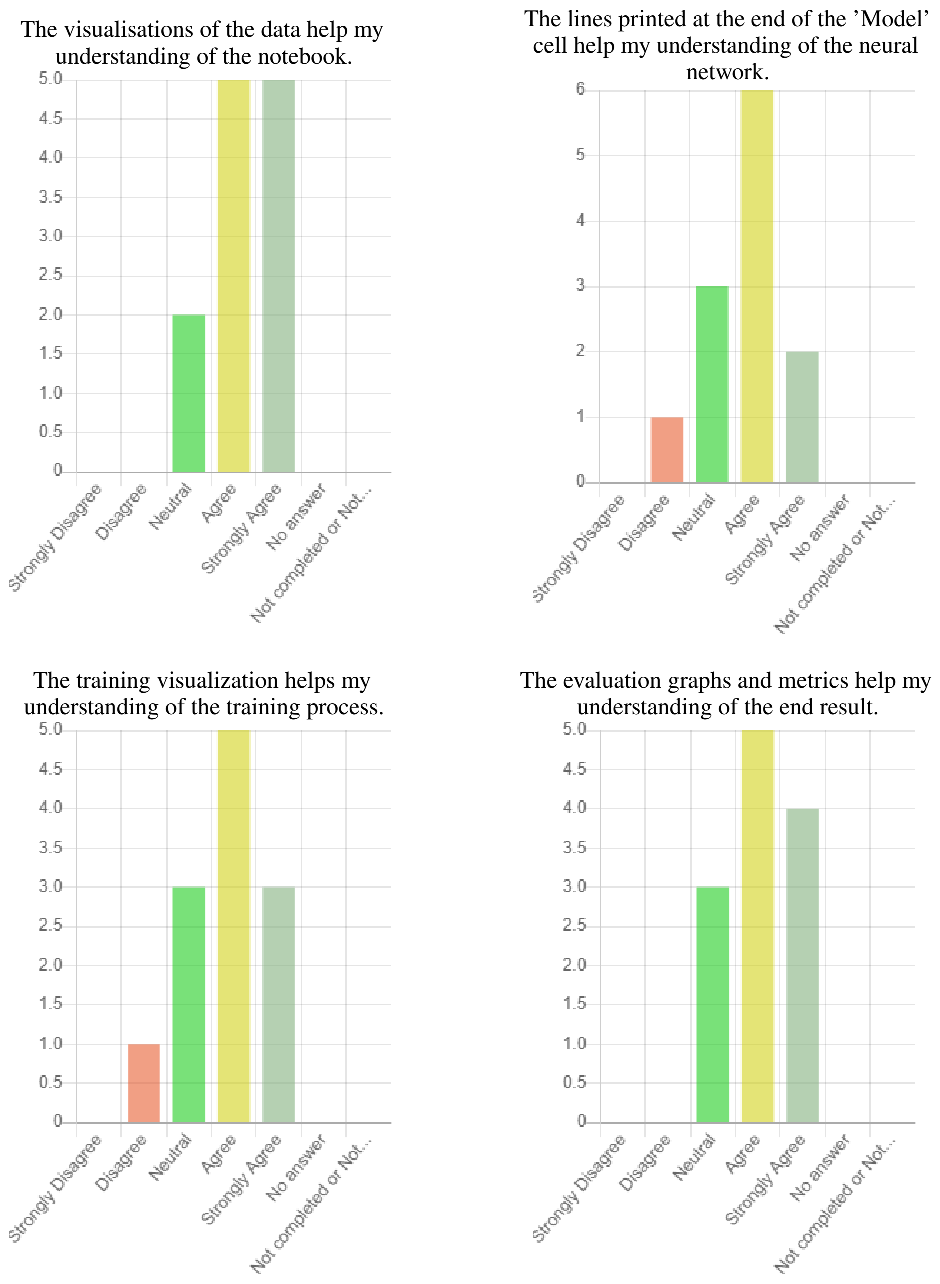
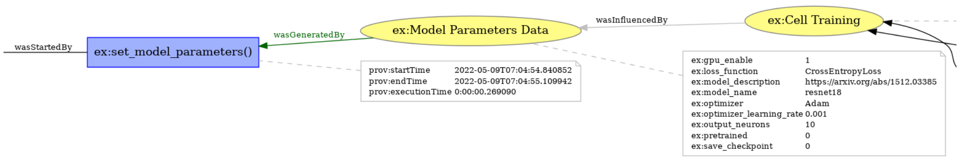


Fig. 7: Ratings of the helpfulness of visual elements in MLProvCodeGen



prov:startTime	2022-05-09T07:04:54.840852
prov:endTime	2022-05-09T07:04:55.109942
prov:executionTime	0:00:00.269090

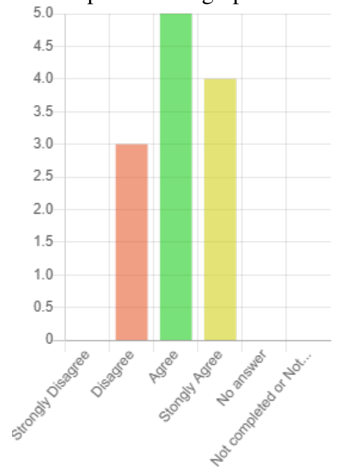
ex:gpu_enable	1
ex:loss_function	CrossEntropyLoss
ex:model_description	https://arxiv.org/abs/1512.03385
ex:model_name	resnet18
ex:optimizer	Adam
ex:optimizer_learning_rate	0.001
ex:output_neurons	10
ex:pretrained	0
ex:save_checkpoint	0

Fig. 8: Excerpt from a generated provenance graph in MLProvCodeGen

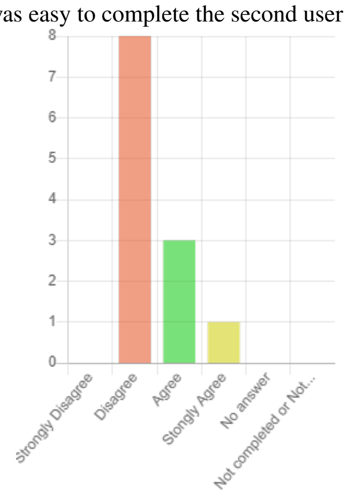
5 Conclusions and Future Work

In this paper, we presented *MLProvCodeGen*, a tool to support the reproducibility of machine learning experiments by integrating provenance data and automatic, template based code generation into one system. We then improved the system by implementing automatic provenance data input, capture, and visualization into the generated notebooks.
We evaluated our system by implementing two use case ML exercises, image classification and multiclass classification, and conducted a user evaluation from which we could draw valuable insights. Future work on *MLProvCodeGen* includes improvements to the provenance graph regarding the way information is presented in the graph, as well as visualization techniques; improving provenance data export by implementing machine readable syntax; as well as expanding the range of ML exercises intergrated into *MLProvCodeGen*.

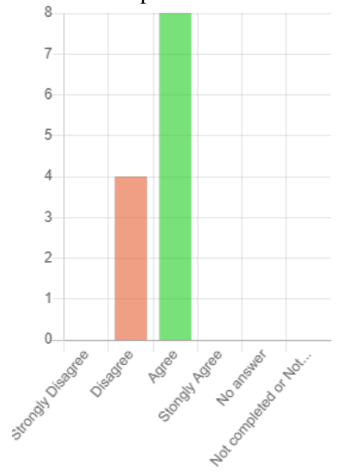
It was easy to find the ’Accuracy’ in the provenance graph.



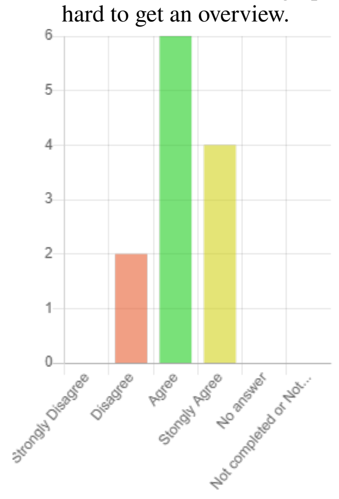
It was easy to complete the second user task.



The data shown in the graph allows me to see the most important parts of the ML experiment.



There is too much clutter in the graph/It’s hard to get an overview.



The relational information shown in the graph allows me to understand how datapoints influence each other.

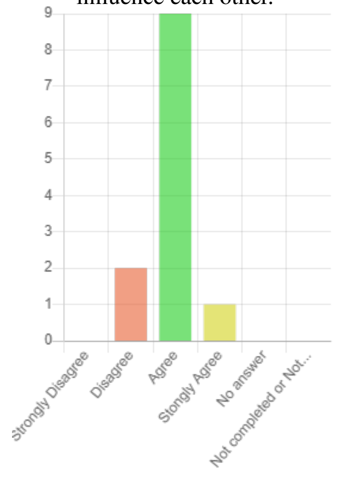


Fig. 9: Ratings of the provenance graph and visualisation in MLProvCodeGen

References

[Ba16] Baker, M.: 1,500 scientists lift the lid on reproducibility. Nature 533/7604, 2016.

[Be13] Belhajjame, K.; B'Far, R.; Cheney, J.; Coppens, S.; Cresswell, S.; Gil, Y.; Groth, P.; Klyne, G.; Lebo, T.; McCusker, J., et al.: Prov-dm: The prov data model. W3C Recommendation 14/, pp. 15–16, 2013.

[Ca06] Callahan, S. P.; Freire, J.; Santos, E.; Scheidegger, C. E.; Silva, C. T.; Vo, H. T.: VisTrails: visualization meets data management. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of data. Pp. 745–747, 2006.

[Ch12] Cheney, J.; Finkelstein, A.; Ludäscher, B.; Vansummeren, S.: Principles of provenance (dagstuhl seminar 12091). In: Dagstuhl Reports. Vol. 2. 2, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[De12] Deng, L.: The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE signal processing magazine 29/6, pp. 141–142, 2012.

[De15] Dey, S.; Belhajjame, K.; Koop, D.; Raul, M.; Ludäscher, B.: Linking prospective and retrospective provenance in scripts. In: 7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15). 2015.

[De19] Deelman, E.; Mandal, A.; Jiang, M.; Sakellariou, R.: The role of machine learning in scientific workflows. The International Journal of High Performance Computing Applications 33/6, pp. 1128–1139, 2019.

[DHS11] Duchi, J.; Hazan, E.; Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research 12/7, 2011.

[Es15] Esteves, D.; Moussallem, D.; Neto, C. B.; Soru, T.; Usbeck, R.; Ackermann, M.; Lehmann, J.: MEX vocabulary: a lightweight interchange format for machine learning experiments. In: Proceedings of the 11th International Conference on Semantic Systems. Pp. 169–176, 2015.

[Fr08] Freire, J.; Koop, D.; Santos, E.; Silva, C. T.: Provenance for computational tasks: A survey. Computing in science & engineering 10/3, pp. 11–21, 2008.

[Fr18] Freire, J.; Koop, D.; Chirigati, F.; Silva, C. T.: Reproducibility using vistrails. In: Implementing Reproducible Research. Chapman and Hall/CRC, pp. 33–56, 2018.

[GNT10] Goecks, J.; Nekrutenko, A.; Taylor, J.: Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome biology 11/8, pp. 1–13, 2010.

[HDB17] Herschel, M.; Diestelkämper, R.; Ben Lahmar, H.: A survey on provenance: What for? What form? What from? The VLDB Journal 26/6, pp. 881–906, 2017.

[He16] He, K.; Zhang, X.; Ren, S.; Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. Pp. 770–778, 2016.

[Hu18] Hutson, M.: Artificial intelligence faces reproducibility crisis. Science 359/ 6377, pp. 725–726, 2018.

[KB14] Kingma, D. P.; Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980/, 2014.

[LCB20] Le, T. H.; Chen, H.; Babar, M. A.: Deep learning for source code modeling and generation: Models, applications, and challenges. ACM Computing Surveys (CSUR) 53/3, pp. 1–38, 2020.

[Le13] Lebo, T.; Sahoo, S.; McGuinness, D.; Belhajjame, K.; Cheney, J.; Corsar, D.; Garijo, D.; Soiland-Reyes, S.; Zednik, S.; Zhao, J.: Prov-o: The prov ontology./, 2013.

[Lu06] Ludäscher, B.; Altintas, I.; Berkley, C.; Higgins, D.; Jaeger, E.; Jones, M.; Lee, E. A.; Tao, J.; Zhao, Y.: Scientific workflow management and the Kepler system. Concurrency and computation: Practice and experience 18/10, pp. 1039–1065, 2006.

[Ma17] Ma, S.; Aafer, Y.; Xu, Z.; Lee, W.-C.; Zhai, J.; Liu, Y.; Zhang, X.: LAMP: data provenance for graph based machine learning algorithms through derivative computation. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. Pp. 786–797, 2017.

[Ma18] Ma, N.; Zhang, X.; Zheng, H.-T.; Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proceedings of the European conference on computer vision (ECCV). Pp. 116–131, 2018.

[Ma20] Ma, L.; Ding, B.; Das, S.; Swaminathan, A.: Active learning for ML enhanced database systems. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. Pp. 175–191, 2020.

[MBC13] Missier, P.; Belhajjame, K.; Cheney, J.: The W3C PROV family of specifications for modelling provenance metadata. In: Proceedings of the 16th International Conference on Extending Database Technology. Pp. 773–776, 2013.

[Mi16] Missier, P.: The lifecycle of provenance metadata and its associated challenges and opportunities. Building Trust in Information/, pp. 127–137, 2016.

[Mo08] Moreau, L.; Freire, J.; Futrelle, J.; McGrath, R. E.; Myers, J.; Paulson, P.: The open provenance model: An overview. In: International provenance and annotation workshop. Springer, pp. 323–326, 2008.

[Mo11] Moreau, L.; Clifford, B.; Freire, J.; Futrelle, J.; Gil, Y.; Groth, P.; Kwasnikowska, N.; Miles, S.; Missier, P.; Myers, J., et al.: The open provenance model core specification (v1. 1). Future generation computer systems 27/6, pp. 743–756, 2011.

[Mo13] Moreau, L.; Missier, P.; Cheney, J.; Soiland-Reyes, S.: PROV-N: The provenance notation./, 2013.

[Na20] Namaki, M. H.; Floratou, A.; Psallidas, F.; Krishnan, S.; Agrawal, A.; Wu, Y.; Zhu, Y.; Weimer, M.: Vamsa: Automated provenance tracking in data science scripts. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. Pp. 1542–1551, 2020.

[OBA17] Olorisade, B. K.; Brereton, P.; Andras, P.: Reproducibility in machine Learning-Based studies: An example of text mining./, 2017.

[Oi04] Oinn, T.; Addis, M.; Ferris, J.; Marvin, D.; Senger, M.; Greenwood, M.; Carver, T.; Glover, K.; Pocock, M. R.; Wipat, A., et al.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20/17, pp. 3045–3054, 2004.

[Pa19] Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In (Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; Garnett, R., eds.): Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035, 2019, URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[Pe11] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12/, pp. 2825–2830, 2011.

[Pi21] Pineau, J.; Vincent-Lamarre, P.; Sinha, K.; Larivière, V.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; Larochelle, H.: Improving reproducibility in machine learning research: a report from the NeurIPS 2019 reproducibility program. Journal of Machine Learning Research 22/, 2021.

[PML20] Parulian, N. N.; McPhillips, T. M.; Ludäscher, B.: A model and system for querying provenance from data cleaning workflows. In: Provenance and Annotation of Data and Processes. Springer, pp. 183–197, 2020.

[Pu18] Publio, G. C.; Esteves, D.; Ławrynowicz, A.; Panov, P.; Soldatova, L.; Soru, T.; Vanschoren, J.; Zafar, H.: ML-schema: exposing the semantics of machine learning with schemas and ontologies. arXiv preprint arXiv:1807.05351/, 2018.

[Ri21] Rieke, J., 2021, URL: `https://github.com/jrieke/traingenerator`.

[Sc17] Schelter, S.; Boese, J.-H.; Kirschnick, J.; Klein, T.; Seufert, S.: Automatically tracking metadata and provenance of machine learning experiments. In: Machine Learning Systems Workshop at NIPS. Pp. 27–29, 2017.

[Sc18] Schelter, S.; Böse, J.-H.; Kirschnick, J.; Klein, T.; Seufert, S.: Declarative metadata management: A missing piece in end-to-end machine learning. Proceedings of SYSML 18/, 2018.

[SK17] Samuel, S.; König-Ries, B.: REPRODUCE-ME: ontology-based data access for reproducibility of microscopy experiments. In: European Semantic Web Conference. Springer, pp. 17–20, 2017.

[SK18a] Samuel, S.; König-Ries, B.: Combining P-Plan and the REPRODUCE-ME ontology to achieve semantic enrichment of scientific experiments using interactive notebooks. In: European semantic web conference. Springer, pp. 126–130, 2018.

[SK18b] Samuel, S.; König-Ries, B.: ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility. In: ISWC (P&D/Industry/BlueSky). 2018.

[SK20] Samuel, S.; König-Ries, B.: Reproducemegit: a visualization tool for analyzing reproducibility of jupyter notebooks. In: Provenance and Annotation of Data and Processes. Springer, pp. 201–206, 2020.

[SK21] Samuel, S.; König-Ries, B.: Understanding experiments and research practices for reproducibility: an exploratory study. PeerJ 9/, e11140, 2021.

[SLK20] Samuel, S.; Löffler, F.; König-Ries, B.: Machine learning pipelines: provenance, reproducibility and FAIR data principles. In: Provenance and Annotation of Data and Processes. Springer, pp. 226–230, 2020.

[So19] Souza, R.; Azevedo, L.; Lourenço, V.; Soares, E.; Thiago, R.; Brandão, R.; Civitarese, D.; Brazil, E.; Moreno, M.; Valduriez, P., et al.: Provenance data in the machine learning lifecycle in computational science and engineering. In: 2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS). IEEE, pp. 1–10, 2019.

[Su13] Sutskever, I.; Martens, J.; Dahl, G.; Hinton, G.: On the importance of initialization and momentum in deep learning. In: International conference on machine learning. PMLR, pp. 1139–1147, 2013.

[SZ14] Simonyan, K.; Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556/, 2014.

[TH+12] Tieleman, T.; Hinton, G., et al.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning 4/2, pp. 26–31, 2012.

[Va17] Van Aken, D.; Pavlo, A.; Gordon, G. J.; Zhang, B.: Automatic database management system tuning through large-scale machine learning. In: Proceedings of the 2017 ACM international conference on management of data. Pp. 1009–1024, 2017.

[Wo13] Wolstencroft, K.; Haines, R.; Fellows, D.; Williams, A.; Withers, D.; Owen, S.; Soiland-Reyes, S.; Dunlop, I.; Nenadic, A.; Fisher, P., et al.: The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic acids research 41/W1, W557–W561, 2013.

[XRV17] Xiao, H.; Rasul, K.; Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747/, 2017.

[Ze12] Zeiler, M. D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701/, 2012.