# A Taxonomy of Tools for Reproducible Machine Learning Experiments

Luigi Quaranta[1], Fabio Calefato[1] and Filippo Lanubile[1]

[1]*Dept. of Computer Science, University of Bari, Via Edoardo Orabona 4, 70125 Bari BA, Italy*

**Abstract**

The broad availability of machine learning (ML) libraries and frameworks makes the rapid prototyping of ML models a relatively easy task to achieve. However, the quality of prototypes is challenged by their reproducibility. Reproducing an ML experiment typically entails repeating the whole process, from data collection to model building, other than multiple optimization steps that must be carefully tracked. In this paper, we define a comprehensive taxonomy to characterize tools for ML experiment tracking and review some of the most popular solutions under the lens of the taxonomy. The taxonomy and related recommendations may help data scientists to more easily orient themselves and make an informed choice when selecting appropriate tools to shape the workflow of their ML experiments.

**Keywords**

Reproducibility, ML experiment, Collaboration

## 1. Introduction

Machine Learning (ML)-based components are today being massively adopted and integrated into traditional software products. The broad availability of ML libraries and frameworks makes the rapid prototyping of ML models relatively easy to achieve. On the other hand, it is challenging to translate ML prototypes into production-ready artifacts, able to offer robust and scalable performances in real-world scenarios [1]. Sato et al. [2] suggest that some solutions to many of the emerging challenges in this area might be built upon consolidated software engineering practices such as Continuous Delivery [3].

A prominent challenge that urges to be addressed is experiment reproducibility. Reproducing an ML experiment means repeating the whole process, from data collection to model building and deployment, requiring – at every stage – multiple optimization steps that must be carefully tracked to ensure reproducibility. Experiment reproducibility is not only a fundamental trait of scientific experimentations, but it is also crucial for many industrial domains, such as banking and automotive, where ML is being employed to solve mission-critical tasks [4].

Ensuring experiment reproducibility can be even more complex in collaborative environments, as teams of data scientists have to safely share not only the code they develop but also the

*AIxIA 2021 Discussion Papers*

luigi.quaranta@uniba.it (L. Quaranta); fabio.calefato@uniba.it (F. Calefato); filippo.lanubile@uniba.it (F. Lanubile)

http://collab.di.uniba.it/luigi-quaranta/ (L. Quaranta); http://collab.di.uniba.it/fabio/ (F. Calefato); http://www.di.uniba.it/~lanubile/ (F. Lanubile)

0000-0002-9221-0739 (L. Quaranta); 0000-0003-2654-1588 (F. Calefato); 0000-0003-3373-7589 (F. Lanubile)

datasets and the experimental environments in which they work. Furthermore, this multifaceted challenge could be exacerbated by the heterogeneity of data science teams, whose members often have different cultural backgrounds and may lack software engineering expertise [5]; in such scenarios, significant training efforts can be required even for the simple adoption of basic software engineering best practices (e.g., code versioning). In the industrial context, the ability to reproduce ML experiments is of paramount importance also to ease the aforementioned transition from an exploratory to a mature production phase, in which ML models are delivered alongside traditional software and carefully monitored over time.

## 2. Taxonomy

Today we are witnessing a multitude of software solutions to support the reproducibility of ML experiments. The choice of the solution that best fits a researcher's or a company's needs is non-trivial, and the abundance of options can be overwhelming at first. To guide potential users in this plethora of available services, we have reviewed some of the most popular solutions and defined a comprehensive characterization of them. We used an inductive approach inspired by Grounded Theory [6], which we found useful to get a sense of this new-born and rapidly-changing area of the software market.

We started with as few preconceptions as possible and strived to build a sound abstraction upon identifying recurring patterns in our reviews of software solutions. We then grouped common ideas into concepts and then concepts into categories, deriving the concise taxonomy shown in Fig. 1. While we cannot claim completeness (we did not try all the available software solutions), we went on examining up to 19 tools until we reached saturation, i.e., we were no more able to identify new tool characteristics through review. Please, refer to the auxiliary online material[1] for a complete list of the tools examined.

The taxonomy groups the tool offering into three main categories, whose corresponding concepts are described in Sections 2.2-2.4.

### 2.1. Tool Sample

To make the taxonomy more evident, we complement it with a selection of four representative software solutions drawn from the tools we reviewed. Before delving into the particulars of our classification, we provide some details on the selected tool sample, as it will be used to exemplify the main ideas throughout the paper.

#### 2.1.1. DVC.

DVC[2] is defined as an open-source version control system for machine learning projects. It offers a command-line interface heavily inspired by `git`. Developers well versed in using `git` get comfortable with DVC; on the other hand, those who have no previous experiences with `git` might find DVC to have a steep learning curve. The implementation also relies on `git`, adding support for cloud storage services and big dataset versioning. As such, DVC offers

[1]https://github.com/collab-uniba/Software-Solutions-for-Reproducible-ML-Experiments
[2]https://dvc.org

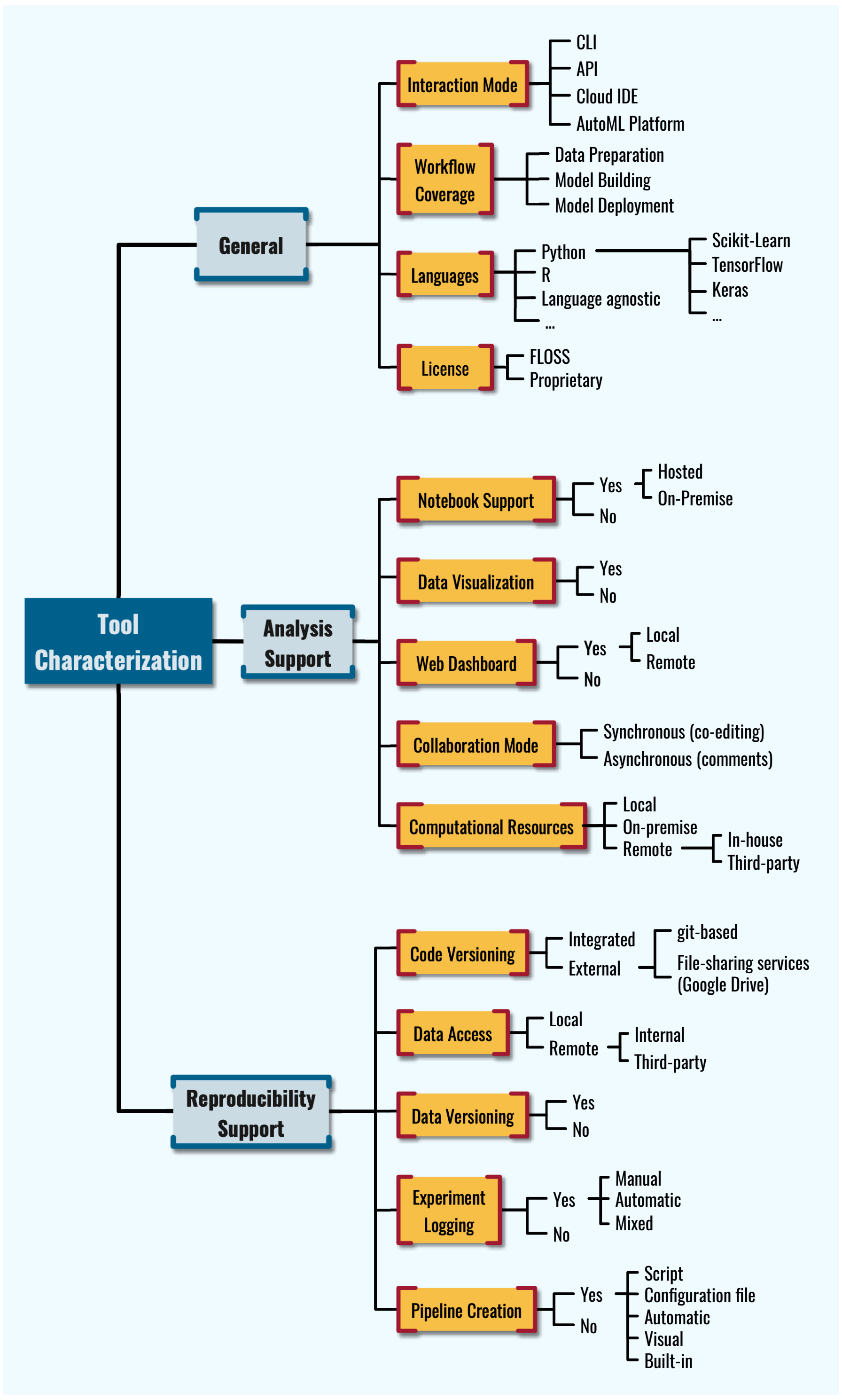

**Figure 1:** The taxonomy inferred from the analysis of existing tools for reproducible ML experiments.

full code and data provenance, enabling users to track the complete history of each artifact they produce. DVC also has experiment management capabilities: experiment metrics can be tracked, and pipelines are automatically detected and represented as a directed acyclic graph (DAG). With all these features into place, experiment runs can be easily repeated and compared. Finally, additional features are push/pull commands that enable model sharing as well as model deployment into production servers.

### 2.1.2. MLflow.

MLflow[3] is an open-source tool that can be used through an API other than as a command-line tool. It is composed of four modules. The *tracking* module is intended for experiment tracking and management, including logging of parameters, code versions, metrics, and output files. Multiple runs can be compared using a web dashboard, which can be either served locally or run remotely (for results sharing). The *projects* module enables experiment reproducibility. Compared to DVC, which automatically tracks data and model provenance, the support for reproducibility appears weaker, as this module relies on conventions to be followed for the experiment to be repeatable. The *models* module offers model deployment support through the definition of a standard format for packaging machine learning artifacts that can be used with a variety of downstream tools (e.g., real-time serving through a REST API). Lastly, *registry* is a module that acts as a centralized model store, featuring model lineage, versioning, and other useful model annotation functions.

### 2.1.3. Google Colaboratory.

Google Colaboratory[4], also known as Colab, is a cloud-based IDE mimicking the popular Jupyter Notebook environment. Users are provided with a generous amount of computing resources. Nevertheless, Colab can be easily wired to external cloud services to afford more computational power. Being built upon Google Drive, Colab offers storage capabilities (including a versioning feature with extended support for notebook diffs), and enables asynchronous collaboration, allowing users to co-edit a notebook and leave comments on its cells. It can also be linked to other Google Cloud Platform services.

### 2.1.4. H2O Driverless AI.

H2O Driverless AI[5] is an AutoML platform that manages the whole end-to-end ML pipeline in complete autonomy. First, the user loads his training data on the platform, sets a few parameters, and starts the training process. Then, H2O Driverless AI seeks the best model within the user's constraints and eventually enables model deployment by creating a REST endpoint or by automatically running the model as a service in the cloud. Users can upload and share extensions in the form of Python recipes, that is, machine learning models, transformers, and scorers specialized for a particular domain (e.g., NLP, time series).

[3]https://mlflow.org
[4]https://colab.research.google.com
[5]www.h2o.ai/products/h2o-driverless-ai

**Table 1**
The tool sample classified according to the features of the General category.

| Tool | Interaction Mode | Workflow Coverage | Languages | License |
|---|---|---|---|---|
| DVC | CLI | All | Language agnostic | FLOSS (Apache 2.0) |
| MLflow | API, CLI | All | Python, R, Java | FLOSS (Apache 2.0) |
| Google Colab | Cloud IDE | Data Preparation + Model Building | Python | Proprietary |
| H2O Driverless AI | AutoML Platform | All | (Python Recipes) | Proprietary |

## 2.2. General

The *General* category captures the basic connotations for comparing the existing software solutions, including how to interact with the tool, the covered phases of ML experiments, the programming languages supported, and the type of software license. As such, this category defines the context in which every further tool feature should be analyzed.

Table 1 exemplifies the classification of our tool selection according to the features covered by this category.

### 2.2.1. Interaction Mode

The existing tools come in different flavors, diverging especially in terms of their *interaction mode*. We identify four main approaches: CLI, API, Cloud IDE, and AutoML Platforms.

**CLI.** Command-Line Interface (CLI)-based tools require users to manually invoke commands to perform data and code versioning, experiment logging, and the tracking of script executions. Sometimes, the commands represent an interface to a local installation of the tool; in other cases, they allow interaction with a remote service. One of the main benefits of a CLI-based solution is that it does not require changes to the source code, and then it can be seamlessly applied to already existing projects. However, CLI-based solutions generally provide less support for computational notebooks and do not offer a centralized platform for team collaboration (e.g., a web dashboard to share experiment results). As such, only a few of the reviewed tools rely exclusively on text-based interaction, as they often combine CLI with one of the other interaction modes.

**API.** Various tools offer an API that requires data scientists to generate a token and add a couple of boiler-plate lines of code to enable automatic logging. Solutions like MLflow and Comet offer a web dashboard that allows easy inspection and comparison of experiment runs and often include other useful visualization capabilities. The API server (along with the webserver hosting the web dashboard, when present) can be installed and executed either locally or - more often - remotely.

**Cloud IDE.** A few software solutions offer a cloud IDE, which generally consists of an implementation of the Jupyter notebook environment, running on a cloud infrastructure. The main advantages are the ease of sharing and portability, as the whole development environment is indeed accessible with a few clicks from whatever connected machine. Cloud IDE services can also support co-editing (e.g., Google Colaboratory) and offer social features, such as the ability to leave document-level and cell-level comments.

**AutoML Platforms.** Automated Machine Learning (AutoML) solutions like H2O and DataRobot typically cover the whole end-to-end pipeline, starting from the collection of raw data all the way through to the model deployment. Experiments conducted using AutoML platforms can be considered reproducible because they are fully automated processes. Yet, AutoML platforms are, of course, very different from the other kinds of tools. Their interaction model is a 'black box' that does not require coding skills for basic usages: users upload a dataset, optionally set some basic parameters in a web-based UI, and wait for the system to search for the best performing algorithm after fully-automatic hyperparameter tuning. Of course, AutoML cannot substitute human expertise, because real-world problems hardly ever fit the simplistic nature of predefined cases. Yet these tools can be useful to teams that are still building their ML expertise, and to experts for a quick assessment of algorithm performance.

### 2.2.2. Workflow Coverage

With the concept of *workflow coverage*, we identify the major phases of an ML experiment, namely *data preparation*, *model building*, and *model deployment*. Most of the tools reviewed offer support for all the major phases of the ML workflow, with the notable exception of Google Colab, which does not provide any explicit support for model deployment.

### 2.2.3. Languages

CLI tools tend to be language-agnostic. Instead, API tools and Cloud IDEs are, for the most part, language-specific. API tools have to be backed by a language-specific library, whereas Cloud IDEs, almost always based on Jupyter notebooks, inherit at least the language limitations of the Jupyter ecosystem, and are often even more restrictive (see Google Colab). The most widely supported languages are by far R and Python, supported by all the reviewed tools. Along with Python comes the support for the major machine learning and deep learning frameworks, such as TensorFlow, PyTorch, Keras, and Scikit-Learn.

**Table 2**
The tool sample classified according to the features of the Analysis Support category.

| Tool | Notebook Support | Data Visualization | Web Dashboard | Collab. Mode | Computational Resources |
|---|---|---|---|---|---|
| DVC | No | No | Yes (remote) | Async (push/pull commands) | Local |
| MLflow | Yes (on-premise) | No | Yes (local) | No | Local + On-premise |
| Google Colab | Yes (hosted) | No | No | Async (co-editing, comments) | Local + Remote (in-house or third-party) |
| H2O Driverless AI | No | Yes | Yes (remote) | No | Remote (in-house or third-party) |

### 2.2.4. License

Another relevant basic aspect characterizing the tools is their license. A fine-grained license classification is out of the scope of this article. We broadly distinguish between FLOSS (Free/Libre Open-Source Software) and proprietary solutions, which we find to be much more common.

## 2.3. Analysis Support

The *Analysis Support* category groups the tool features designed to help data scientists with their analytical tasks, including collaborative data analysis and data modeling tasks.

In Table 2, we report a classification of our tool sample with reference to the features covered by this category.

### 2.3.1. Notebook Support

During their everyday work, data scientists make use of many tools. Among these, *computational notebooks* play a leading role in enabling fast and interactive prototyping while offering a lightweight and integrated form of experiment documentation. Moreover, computational notebooks facilitate collaboration among stakeholders, allowing professionals to report analytical results to diverse kinds of audiences effortlessly.

Computational notebooks enable the so-called ‘computational narratives,’ i.e., the embedding of complex computations into easy-to-follow narratives that make notebooks clearer than bare source code, especially with the intrinsic complexity of ML scripts. Jupyter Notebook and its direct descendant, JupyterLab, are by far the most widely-adopted computational notebook systems [7]. Given the enormous spread of such tools, many of the software solutions for reproducible ML experiments have been built around them or at least designed to fully support their daily use.

With notebook support, we intend the possibility to use the main features of a tool from within a computational notebook. In general, because they are operated through simple library functions calls, API tools can be leveraged from scripts as well as notebooks. For the same reason, notebooks lend themselves well to be adopted with Cloud IDEs. On the contrary, CLI tools do not offer native support for computational notebooks (unless they are treated as scripts, leveraging external utilities such as `nbconvert`). When notebook support is present, we distinguish the case in which notebooks are executed on local resources (on-premise) from that in which notebook instances are hosted on cloud resources (as in the case of Google Colab).

#### 2.3.2. Data Visualization

The concept of *data visualization* refers to the availability of built-in data visualization features, which are accessible from the tool GUI. Features that merely rely on language-specific plotting frameworks (e.g., matplotlib) and require users to write code do not qualify. We have found this to be quite a rare feature offered exclusively by AutoML platforms, which tend to provide users with data visualizations proactively, in order to guide them in the choice of basic AutoML parameters.

#### 2.3.3. Web Dashboard

Many solutions (especially API tools) offer an integrated web dashboard to show records of experiment runs. Often such dashboards automatically display summary plots and statistics that enable immediate evaluation and comparison of experimental results. Web dashboards can be hosted on a local or remote server. In the latter case, they allow data scientists to share results with the team in real-time.

#### 2.3.4. Collaboration mode

*Collaboration mode* refers to the presence and type of collaborative features supported. When available, we distinguish between *synchronous*, indicating co-editing capabilities, and *asynchronous*, covering mainly the ability to leave comments and reactions to the work of other colleagues, and the presence of push/pull commands to share content. Synchronous and asynchronous collaboration modes appear to be quite rare.

#### 2.3.5. Computational Resources

With the growing popularity of the SaaS delivery model, it is not surprising that many of the software solutions reviewed are powered by remote computational resources. These cloud resources can be either in-house, i.e., offered by the same company that provides the ML tool, or third-party, i.e., supplied by an external cloud-computing provider such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform. In some cases, tools are designed to be

**Table 3**
The tool sample classified according to the features of the Reproducibility Support category.

| Tool | Code Versioning | Data Access | Data Versioning | Experiment Logging | Pipeline Creation |
|---|---|---|---|---|---|
| DVC | Yes (external, git-based) | Local + Remote (third-party) | Yes | Yes (manual) | Yes (automatic) |
| MLflow | Yes (external, git-based) | Local + Remote (third-party) | No | Yes (hybrid) | Yes (config. file) |
| Google Colab | Yes (file-sharing services - Google Drive) | Remote (internal or third-party) | Yes | No | No |
| H2O Driverless AI | Yes (integrated) | Remote (internal or third-party) | Yes | Yes (automatic) | Yes (built-in) |

executed locally on the user's machine. In other cases, they can be deployed on-premise (e.g., on company servers).

## 2.4. Reproducibility Support

The category of *Reproducibility Support* groups the features that reinforce the repeatability of ML experiments.

To ensure reproducibility, it is essential to track both the operations that are performed during each phase of the ML workflow and the artifacts that are produced after each operation. Indeed, an ML workflow is usually represented in the form of a pipeline, i.e., a sequence of processing steps in which the output of one step becomes the input of the next.

To ensure end-to-end reproducibility of an ML experiment, it is necessary to have access to the specific version of data that was used when the model was trained in the first place, along with the specific version of code. Moreover, the frameworks used to build ML models often depend on complex development environments that are hard to reproduce too. Hence, for ML experiments to be fully repeatable, dedicated software should support the versioning of the entire working environment. This is a complex matter; nevertheless, during the last few years, modern containerization technologies are making it easier to accomplish.

In Table 3, we report the classification of our tool sample according to the features pertaining to this category.

### 2.4.1. Code Versioning

Code versioning is an established practice of software engineering. Code versioning in ML can be a feature integrated with the software solution for experiment reproducibility or within an external tool; in the latter case, it either leverages git-related technologies (e.g., GitHub or the local git installation itself, as in the case of DVC) - or is based on other services such as Google Drive, which often come with their built-in versioning technology.

### 2.4.2. Data Access and Versioning

Although versioning data may seem like a natural extension of code versioning, it comes with its peculiar challenges. Data requires much more space to be stored, and deltas can be harder to represent. Furthermore, versioning data may entail providing detailed information about the full provenance of data, starting from the original source all the way through the pipeline processing steps.

Ensuring access to the specific version of data involved in a model training session entails a two-fold problem: (1) *data access*, i.e., ensuring that data is somehow made available to the team, which is not always the case, especially when a dataset is stored locally in the experimenter machine and never backed-up in a cloud storage service; (2) *data versioning*, i.e., ensuring that the data is versioned every time it gets processed. Many software solutions guarantee the possibility to store data remotely, leveraging cloud storage services. Some of them support data versioning, sometimes in lightweight flavors (e.g., tracking of SHA digests), sometimes offering complex versioning systems that track the full data provenance/lineage.

### 2.4.3. Experiment Logging

Model building generally requires many experiment runs based on a cyclic process of data cleaning, feature engineering, and hyperparameters tuning in which plenty of decisions are made. In the case of unsatisfying results, decisions can be revised, resulting in a new experiment run. Thus, the number of experimental iterations can quickly escalate, making it difficult to keep track of the results without proper tooling. Most of the software solutions reviewed offer experiment logging. Data scientists can log relevant information about the specific experiment run, such as dataset sources and versions, project dependencies, hyperparameters, data visualizations, and metrics. This allows retrieving the rationale behind each decision and enables the comparison across the experiment runs.

Tools supporting this feature usually demand manual interventions by the user, requiring CLI commands to be issued or the invocation of API functions in the scripts. In some cases, logging can be automatic (e.g., Driverless AI) or hybrid (e.g., MLflow, Neptune). Ultimately, experiment logging can be further enriched with automatic recording of system performance stats.

### 2.4.4. Pipeline Creation

Last but not least, pipeline creation is a fundamental concept that - together with the others collected in this category - enables the actual reproducibility of ML experiments. A *pipeline* can be defined in many ways. Sometimes the pipeline is automatically detected (as in the case of

DVC); otherwise, users need to define it imperatively via scripts (e.g., Spell.run), declaratively via configuration files (e.g., MLflow), or even visually (e.g., RapidMiner).

## 3. Conclusions

We reviewed a large and varied set of tools for reproducible ML experiments. The landscape for these tools is right now very active – new alternatives keep popping out, and key features missing from one solution might just be added to it as one reads. Still, by highlighting the key features, the taxonomy can be used to narrow down the search space to a few candidate solutions and draw some guidelines for final selection.

*For data science teams with good software engineering experience, consider using a CLI-based solution*, such as DVC. The familiarity with `git` and its push and pull model may drastically decrease the learning curve; plus, CLI-based solutions are very flexible and easy to fit within a pre-existing infrastructure via shell scripting.

*If the crucial feature is the graphical comparison of experiment runs, consider using an API-based solution*, such as MLflow. In general, API-based solutions come with a web dashboard that makes it easier and more pleasant to check for the optimal configurations. API-based solutions are also easy to integrate and 'just work' whenever one can make their code run.

*If no solution checks all the needed boxes, consider adopting more than one.* While the reviewed solutions are competitors, their approach to ML experiment reproducibility can vary drastically, to the point that they can co-exist in a work environment. For example, one can use DVC for managing the reproducible pipeline in combination with MLflow for logging and graphically comparing results across each experiment run. With so many potential tool combinations, the sky is the limit.

Our GitHub repository provides environment templates to speed up the configuration process as well as a realistic case study that was taken from Kaggle to start fiddling with the tools.

## References

[1] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, D. Dennison, Hidden technical debt in machine learning systems, in: Advances in neural information processing systems, 2015, pp. 2503–2511.

[2] D. Sato, A. Wider, C. Windheuser, Continuous Delivery for Machine Learning - Automating the end-to-end lifecycle of Machine Learning applications, 2019. URL: https://martinfowler.com/articles/cd4ml.html.

[3] J. Humble, D. Farley, Continuous delivery: reliable software releases through build, test, and deployment automation, Pearson Education, 2010.

[4] M. Hutson, Artificial intelligence faces reproducibility crisis, Science 359 (2018) 725–726. URL: https://www.sciencemag.org/lookup/doi/10.1126/science.359.6377.725. doi:10.1126/science.359.6377.725.

[5] M. Kim, T. Zimmermann, R. DeLine, A. Begel, The Emerging Role of Data Scientists on Software Development Teams, in: Proceedings of the 38th International Conference on Software Engineering, ACM, 2016, pp. 96–107. Tex.ids: kimEmergingRoleData2016a.

[6] K.-J. Stol, P. Ralph, B. Fitzgerald, Grounded theory in software engineering research: a critical review and guidelines, in: Proceedings of the 38th International Conference on Software Engineering, ACM, Austin Texas, 2016, pp. 120–131. URL: https://dl.acm.org/doi/10.1145/2884781.2884833. doi:10.1145/2884781.2884833.

[7] J. M. Perkel, Why Jupyter is data scientists' computational notebook of choice, Nature 563 (2018) 145–147.