A Large-Scale Study of ML-Related Python Projects

Samuel Idowu
Chalmers | University of Gothenburg
Sweden

Yorick Sens
Ruhr University Bochum
Germany

Thorsten Berger
Ruhr University Bochum and
Chalmers | University of Gothenburg
Germany | Sweden

Jacob Krüger
Eindhoven University of Technology
The Netherlands

Michael Vierhauser
Ruhr University Bochum
Germany

ABSTRACT

The rise of machine learning (ML) for solving current and future problems increased the production of ML-enabled software systems. Unfortunately, standardized tool chains for developing, employing, and maintaining such projects are not yet mature, which can mainly be attributed to a lack of understanding of the properties of ML-enabled software. For instance, it is still unclear how to manage and evolve ML-specific assets together with other software-engineering assets. In particular, ML-specific tools and processes, such as those for managing ML experiments, are often perceived as incompatible with practitioners' software engineering tools and processes. To design new tools for developing ML-enabled software, it is crucial to understand the properties and current problems of developing these projects by eliciting empirical data from real projects, including the evolution of the different assets involved. Moreover, while studies in this direction have recently been conducted, identifying certain types of ML-enabled projects (e.g., experiments, libraries and software systems) remains a challenge for researchers. We present a large-scale study of over 31,066 ML projects found on GitHub, with an emphasis on their development stages and evolution. Our contributions include a dataset, together with empirical data providing an overview of the existing project types and analysis of the projects' properties and characteristics, especially regarding the implementation of different ML development stages and their evolution. We believe that our results support researchers, practitioners, and tool builders conduct follow-up studies and especially build novel tools for managing ML projects, ideally unified with traditional software-engineering tools.

KEYWORDS

machine learning, ML-enabled systems, evolution, mining study, open-source projects, large-scale study, TensorFlow, scikit-learn

ACM Reference Format:
Samuel Idowu, Yorick Sens, Thorsten Berger, Jacob Krüger, and Michael Vierhauser. 2024. A Large-Scale Study of ML-Related Python Projects. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24), April 8–12, 2024, Avila, Spain.* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3605098.3636056

1 INTRODUCTION

Developing machine-learning (ML)-enabled software [2] differs from traditional software development. Specifically, developing an ML model (i.e., traditional ML or deep learning) involves non-linear, iterative, and exploratory experimentation to determine acceptable models for specific requirements [29]. Such experiments are essential for developing ML-enabled systems, but pose novel challenges to developers [3, 4, 17]. For instance, traditional software engineering tooling provides limited support for managing ML assets, due to the diverse types of assets and the iterative, intuition-based exploration of the solution space typically associated with ML experiments [3, 10, 17, 29]. To this end, novel tools and further research—e.g., strongly demanded by Arpteg et al. [4]—is needed to effectively and efficiently develop high-quality and production-ready ML-enabled software. One example of an existing gap between traditional software-engineering and ML development practice is the use of version-control systems for managing the evolution of assets. While such systems (e.g., Git), are de facto standard tools in software engineering, they are generally ill-equipped for managing ML assets, such as datasets and binary model files. However, managing evolution is especially important for exploratory ML experiments, which may involve a large number of these assets, but also for production-oriented software systems, in which ML-models and related source code interact with other software assets. As such, current software-engineering methods and tools lack support for ML development at the right level of abstraction. For example, one cannot query specific ML assets based on their performance or the features used.

Many different tools have been proposed to address such gaps in managing ML-specific assets, including workflow management, pipeline management, model management, dataset management, and experiment management [19, 28, 32] tools. The latter, for instance, offer means to track assets and processes during and after ML-experiment iterations, supporting reproducibility [6, 20, 34], collaboration [40], or traceability [23]. While such tools are becoming increasingly popular, they have not yet fully matured [23] and are especially not well integrated with common software-development tooling. For instance, many such tools target data scientists who work rather independently, lacking support for the collaboration required in large-scale industrial software development. Furthermore, integration with existing, well-established development tools, such as version-control systems, is essential to facilitate adoption in practice.

To build better tools for managing ML assets, we need to improve our empirical understanding of ML-related project development and the common *properties*, *asset types*, *development stages*, and *transitions* between these stages that are involved. Recent studies have investigated the features and support offered by emerging ML asset management tools [13, 17–20, 28, 32, 37] and attempted to characterize ML experimentation empirically [5, 8, 16, 29, 33].

Unfortunately, the body of knowledge on the characteristics and practices related to real ML-related projects is sparse. Early studies exist on specific aspects, such as code styles, on collaboration practices, often using small datasets of real projects. Large-scale studies on larger datasets, investigating general characteristics, identifying the exact stages, and also the history of changes, are still missing. For instance, what types of projects are currently developed? How do these projects and their assets evolve? How do developers manage and transition between ML development stages?

We contribute in this direction and present a large-scale mining study. We provide a dataset of 31,066 ML-related projects on GitHub and collected insights on their types, characteristics, and evolution. Our focus was on Python as the most popular language for ML-related projects, as well as on projects relying on the two most popular ML libraries TensorFlow and/or scikit-learn. We formulated the following research questions:

RQ$_1$ *What types of ML-related projects are maintained on GitHub?* First, we were interested to learn what types of projects are ML-related. We manually analyzed a smaller, random sample of the whole dataset, to identify and define the different types of projects we found, and to learn about their prevalence.

RQ$_2$ *Which development stages can be found in ML-related projects on GitHub?* Second, we aimed at a better understanding of which stages of ML development [9, 17] are maintained in repositories. For example, some repositories may involve data processing or modeling stages only, while others may build on pre-trained models to focus only on the prediction stage. Our results provide insights into typical development stages and how developers manage their ML-enabled projects.

RQ$_3$ *How do ML projects on GitHub evolve and which practices are applied?* Third, we quantitatively analyzed how typical ML-enabled projects evolve, focusing on the different stages identified. For example, the ML model, or its performance metrics, may be constantly updated in the repository to ensure traceability, or they may be kept as they are to save space.

In summary, we contribute a large-scale dataset and empirical data on the projects. All artifacts related to this paper, including the lists of projects, scripts, and extended results, are available in our online appendix.[1] As such, we contribute to the general understanding of how ML-related projects are developed and how Git is used to maintain their assets and evolve these projects. We believe that our results can help researchers, practitioners, and tool builders improve software-engineering and ML development practices.

2 BACKGROUND AND RELATED WORK

We now introduce the necessary background on ML model development processes, as well as we discuss the literature that is related to our study.

[1]https://github.com/isselab/2024-appendix-mlprojectmining

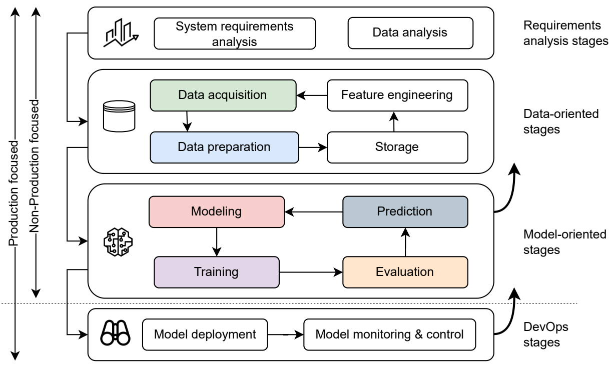

Figure 1: ML development stages [17]; the colored ones are those actually found in real projects in prior studies [9].

2.1 Background

Like traditional software-engineering processes [31], which include design, coding, testing, deployment, and maintenance, ML development follows processes grounded in data science and data mining.

Figure 1 shows a process for developing ML projects, adapted from Idowu et al. [17] and Biswas et al. [9], including requirements-analysis, data-oriented, model-oriented, and DevOps stages [3, 4]. Building on the data, the model-oriented stages include modeling, training, evaluation, and prediction of the ML model. We also show that ML-enabled projects can be production-focused or non-production-focused. For example, ML for research papers is typically non-production focused and does not require DevOps. In contrast, industrial ML-enabled projects become production-focused when the company decides to leverage the models in products [17]. Other processes are, for instance, CRISP-DM [38], KDD [12], and TDSP [21].

Following Biswas et al. [9] we focused on six stages commonly found in ML pipelines of real projects, as highlighted in Fig. 1:

- *Data Acquisition* focuses on data quality, completeness, consistency, and relevance of the raw data being collected from the identified sources.
- *Data Preparation* is typically tedious and time-consuming, often requiring substantial resources and human expertise to process the acquired raw data into feasible structures. Some preparations can be performed automatically using Extract-Transform-Load (ETL) tools, which can process data (e.g., loading, formatting, and outputting in a data warehouse).
- *Modeling* includes model planning and selection, and data mining to discover essential properties of the data relevant for the model.
- *Training*, using selected features and labeled data, follows the modeling stage and focuses on optimizing the model performance via multiple iterations of (hyper-)parameter search.
- *Evaluation* is concerned with evaluating a model via specified metrics, such as accuracy and latency performance, which should be done on real-world data to assess the model's performance under production and non-production conditions.
- *Prediction* involves using the trained ML model on unseen examples (i.e., unlabeled data). The prediction capability of a model can also be evaluated by comparing its performance using test and training datasets.

2.2 Related Work

Researchers have mined open-source ML projects from GitHub [5, 7, 9, 14, 27, 33, 35] before. Bhatia et al. [7] present a qualitative and quantitative empirical study on the contributions and degree of collaboration in ML-enabled system. They mined over 1,300 ML GitHub repositories and over 67,000 forks. Gonzalez et al. [14] also study ML tools and application repositories hosted on GitHub to identify their unique properties, development patterns, and trends. Their work provides a detailed study of developer workflow, measuring their collaboration and autonomy within a repository. Similarly, Biswas et al. [9] study the pipelines of 21 Python-based matured data science systems from GitHub. In addition to GitHub repositories, they also mine 105 data-science projects from Kaggle. Their study explores and identifies the typical stages of ML pipelines, how they are connected, and the differences between pipelines used in practice and those from theory. Van Oort et al. [35] mined 74 open-source ML-enabled systems to discover code smells and refactoring opportunities in their source code. In addition, they observed multiple factors affecting the maintainability and reproducibility of ML-enabled systems. Simmons et al. [33] performed a large-scale empirical study of coding standards in ML-enabled systems to investigate their adherence to code standards. The study compared over 1,000 open-source ML-enabled systems to similarly-sized non-ML systems with similar quality and maturity levels. Barrak et al. [5] mined repositories to explore the degree of coupling between ML-specific and other software assets, as well as the adoption of ML versioning features. They empirically studied 391 ML-enabled systems on GitHub that were managed using DVC. Nahar et. al. [27] contribute a dataset of 262 ML products from GitHub and manually analyze 30 of these. They report about collaboration within interdisciplinary development teams, software architecture related to ML models, development processes, testing, operation, and the use of responsible AI.

Similar to these related works, we mined software repositories to collect ML-related projects from GitHub to identify properties of ML-related projects' development stages and assets. Our study mainly deviates from the previous ones by the size of the analyzed dataset; by the range of project types covered; and by our large-scale analysis of the ML stages present in projects, and especially our investigation of the history. At the same time, we were inspired by these works, such as by Simmons et al. [33], who conclude that ML codebases are distinct from traditional SE codebases and do not follow traditional SE conventions; or by Biswas et al. [9], whose results (i.e., presence of ML stages in real project codebases) and published artifacts we build upon to identify ML stages in our repositories.

3 METHODOLOGY

We now describe our data collection procedure, as well as methods to analyze the resulting dataset and answer our research questions.

3.1 Data Collection

Like prior studies on traditional software development [15, 39], our primary data source was real public repositories mined from GitHub. Our inclusion (IC) and exclusion criteria (EC) were as follows:

IC_1 The project is implemented in Python.

IC_2 The project depends on one or both of the established ML libraries scikit-learn or TensorFlow.

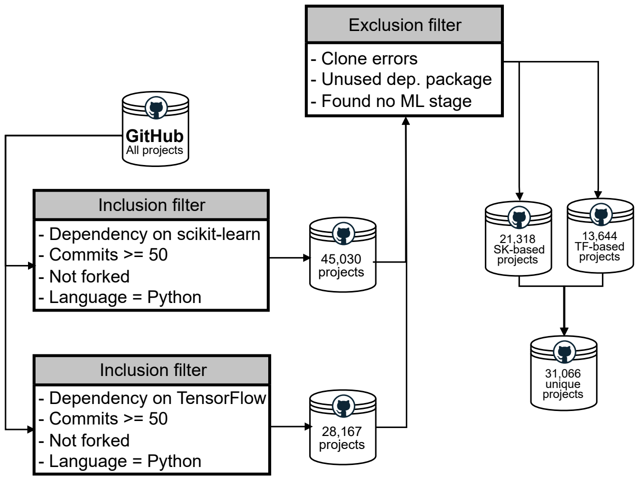

Figure 2: Selection process for GitHub repositories

IC_3 The project is original, meaning that it is not forked from another repository.

IC_4 The project repository involves at least 50 commits.

EC_1 The project causes errors when cloning.

EC_2 The project is not using one of the two ML libraries.

EC_3 The project has no identifiable or established stages of the ML workflow when mapped with our library API dictionary (explained in Sec. 3.2).

Figure 2 illustrates our mining process. We focused on projects developed primarily in Python (IC_1) as the most popular language for ML-related projects [9, 14, 30]. Restricting the study to one programming language makes the projects comparable, yielding higher internal validity of our analysis. We used the GitHub dependency graph to identify projects with a dependency on at least one of the two most popular ML development libraries (IC_2): scikit-learn and TensorFlow [1, 14, 22, 30]. Scikit-learn is arguably the most common library used for classical ML projects, while TensorFlow is a standard library for DL projects. We identified 223,822 GitHub repositories that depend on scikit-learn and 130,580 that depend on TensorFlow. After removing forks from our dataset (IC_3), we were left with 221,084 and 128,084 repositories for each library. This was done because we considered that they would not provide any additional insights but rather skew the results, as large projects with many forks would otherwise be counted disproportionately. In fact, the analysis of fork ecosystems for ML-related projects would constitute a study of its own. Since we were interested in the development process of ML-related projects, we considered only projects with a somewhat substantial evolution history for our analysis (IC_4). To that end we made a reasonably subjective decision to filter out projects with fewer than 50 commits, leaving 45,030 and 28,167 projects. After merging the two lists we were left with 61,062 unique repositories after removing 12,135 duplicates that depend on both libraries.

We then cloned all remaining unique repositories for further analysis of the source code, excluding those that we could not clone due to errors (EC_1), such as HTTP request failures and non-existent Git remote reference errors. This removed a marginal number of repositories. Since the GitHub dependency graph does not guarantee that a project listing a particular library actually uses that library by invoking the corresponding API, we filtered out projects that do not invoke any of the two libraries' APIs in any of the code files in their repository (EC_2). We performed this filtering by analyzing the import directives of the Python source files. To this end, we transformed all Jupyter Notebooks into Python source files. Thereafter, we created abstract syntax trees for all Python files from which we extracted a list of imported modules and skipped files that we could not automatically process due to syntax errors. Finally, we checked whether one of the two libraries or any of their modules were in the resulting list of each project.

Using the ML development stages (cf. Fig. 1) identified for GitHub repositories by Biswas et al. [9], we filtered out (cf. Sec. 3.2) projects that do not involve any of these (EC_3). After this selection, we obtained 21,318 scikit-learn-based projects, 13,644 TensorFlow-based projects, of which 3,896 projects use both libraries, leaving us with 31,066 projects as subjects for our empirical analysis.

3.2 Data Analysis

Identifying project types (RQ_1). We were first interested to understand what types of projects are ML-related. To characterize the dataset, we conducted a manual analysis on a randomly selected sample of 100 repositories. We defined the categories during this process, as we observed recurring project types. Because it is hard to clearly differentiate these categories, we assigned some projects to multiple categories. We manually reviewed the projects, also documenting using what strategy and using what parts of the projects we could identify the project types.

Identifying ML Stages (RQ_2). To identify concrete ML development stages in our subject projects, we used the API dictionary defined by Biswas et al.[9], which maps popular ML library calls from source code to the development stages. A similar API-dictionary method has also been used by Wang et al. [36] when they analyzed external dependencies used in computational notebooks written in Python. In addition to scikit-learn and TensorFlow, the API dictionary also contains functions from additional libraries involved in ML development, such as pandas, numpy, keras, theano, and caffee. So it covers the most popular ML libraries studied in prior work [1, 22, 30]. Since a project can use multiple ML libraries, we found it useful to consider these other libraries for the API dictionary beyond the two we used for our selection process.

Before the actual analysis, we transformed all Jupyter Notebooks into normal Python source files, which was technically required for the following steps. For each Python source file (and converted Jupyter Notebook) in a given repository, we parsed the code into an abstract syntax tree and iterated over all of its elements, extracted function calls, and mapped them to ML stages using the dictionary. With this method, we extracted a list of ML stages implemented in each file and from that created a list of files implementing each stage for the entire project. For this specific RQ we applied the technique only on the most recent state of the main branch. Based on this information we investigated which stages are present in each project and how many files are associated with those. We also counted how often multiple stages were combined in a single file and which stages frequently occur together.

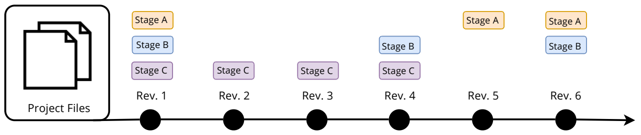

Figure 3: Illustration how we determined the changed ML stages per commit

Evolution of ML Stages (RQ_3). We observed the delta between successive commits for each project to investigate how it evolved over time. For this purpose, we employed the previously described method for the identification of ML stages on the files changed in each commit to determine which stages of ML workflow were affected. Specifically, for each commit, we iterated through all changed source files and applied the API mapping to both the old and the new source code of the file. We chose to analyze the entire file rather than only the changed lines since the source code is very interconnected. Therefore, it is difficult to assign individual lines to ML stages. For example, if a line defining some hyperparameter was changed, and that parameter is used later in the code when a corresponding ML function is called, this would normally be considered a change affecting the corresponding ML stage, although the changed line does not contain any functions from the API dictionary. Consequently, we considered every change in a file implementing a specific ML stage as a change to that stage. Figure 3 illustrates how each commit's affected ML stages were determined. During our analysis, we only considered the main branch for each project.

To understand the development process of ML-related projects, we investigated how often changes related to each stage were done throughout the lifecycle of the projects. To this end, we observed the file changes and the corresponding affected ML stages for each project; We then calculated the ratio of projects that introduced changes in specific stages per commit, over the entire lifecycle of all the projects studied. We also investigated the proportion of the projects' commits affecting each ML stage across projects of different sizes. We grouped the repositories into five groups based on commit count:

Group A:	7,324 projects with (50,70] commits.
Group B:	6,107 projects with (70,100] commits.
Group C:	5,420 projects with (100,150] commits.
Group D:	5,843 projects with (150,300] commits.
Group E:	5,682 projects with (300,113K] commits.

We used this grouping to balance the number of repositories and the extent of their evolution within as well as between groups.

4 RESULTS

We now present and discuss the results for our research questions.

To generally characterize the projects, Table 1 and Fig. 4 summarize core statistics of the 31,066 projects in the whole dataset. The identified ML-related projects span a wide range of scales, from small to very large projects, as follows. The vast majority of the projects are rather small with a median of 110 commits, 2 contributors, 2 branches, 1 star, and 26 source files. The average number of ML-related source files (source files that apply one of the ML development stages defined in Sec. 3.1) is 15, so roughly half of the source files implement ML components of the software.

Table 1: Core characteristics of all 31,066 projects in our dataset

	min	max	mean	median
number of source files	0	17,747	75	26
number of ML files	0	5,318	42	15
number of commits	50	27,692	304	110
number of contributors	1	459	9	2
number of branches	1	1,601	6	2
number of stars	1	131,686	69	1

4.1 Types of Projects (RQ_1)

In our manual project analysis, we observed that the most helpful assets were the project documentation (README, etc.), but also the project structure and types of files present. In total, we identified 7 distinct categories of projects in our random sample of 100 repositories.

4.1.1 Identified Types. In the following we define the identified types of projects and provide an example for each category. Figure 5 shows the prevalence of the different categories.

Experiment. This ML project type refers to projects that aim to develop a suitable ML model for a specific application by conducting multiple runs of training, testing and validation using variations of the hyperparameters, data features, and training procedure. The results of each run usually contain a trained model, its performance metrics and its predictions on the dataset. Incidentally we noticed that most experiments appear not to store the results from multiple experiment runs, only committing the latest version of the trained model and associated source code.

Example: An example of a project with ML experimentation is "Photon Sphere."[2] The goal of this project is to filter pernicious DNS requests using a DNN. It includes model implementations, scripts for training them and multiple trained models as binary files. Some commits make small changes to the source code for training the models, especially modifying hyperparameters. Others change the model's binary files, as the developers trained new model versions. Such commits can be related to experiment runs. Interestingly, the developers do not store multiple versions of the same model in the repository, but rather replace the binary files as better performing models become available. Neither do they store results from experiment runs, like performance metrics or the hyperparameters used. The different models in the repository are apparently different types of models for different purposes.

Education. This ML project type includes all content that supports education, like practical examples for university courses, homework solutions or student projects. It is usually explicitly stated in the project description for which course they were developed.

Example: An example of an education project is "Machine Learning Engineer Nanodegree."[3] This is part of the material for a university course. It contains five projects, each with a task description, data and template code. The projects are typical problems to be solved with ML, like predicting Titanic survivors.

Tutorial. This type of ML-related project is designed to facilitate understanding of a specific ML topic. This category shares similarities with with education, but with the difference that tutorials are not restricted to formal types of education (e.g., university education), but are often less formal and target practitioners; they can also be intended for self-study.

Example: A tutorial example is "Putting TensorFlow Models Into Production,"[4] describing model serving with TensorFlow. It contains multiple Jupyter notebooks with explanations and examples.

Research. This type refers to projects that accompany research papers. The aim is usually the development of new ML models, training methods, optimization techniques or applications of ML. Experiments are often conducted as part of these projects, leading to most research projects also being classified as experiments. Some research projects could also be called "system prototypes." We distinguish them from systems by the fact, that the main goal of research projects is the accompanying scientific publication, not a possible usage by an end user.

Example: A research project example is "UCADI."[5] It is developed as part of a research paper to provide a framework for training a model to automatically diagnose COVID-19. The system prototype consists of a central sever that trains the model based on data obtained from multiple clients (hospitals using the system) while preserving data privacy.

[2]https://github.com/jkerrigan/photon_sphere

[3]https://github.com/italopguimaraes/Machine-Learning-Engineer-Nanodegree

[4]https://github.com/bugra/putting-tensorflow-models-to-production

[5]https://github.com/HUST-EIC-AI-LAB/UCADI

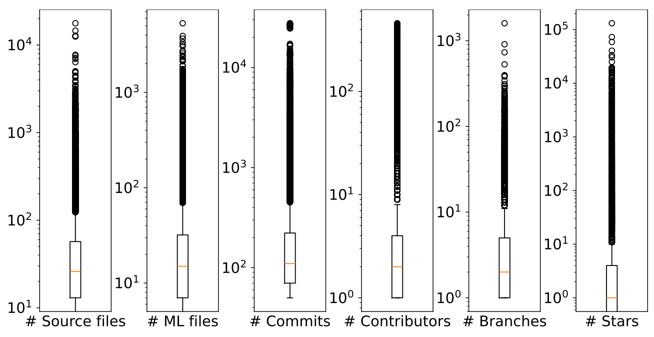

Figure 4: Distributions of core characteristics of the 31,066 projects visualized as boxplots

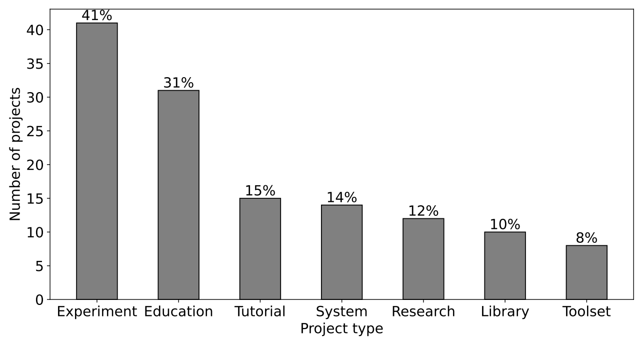

Figure 5: Number of projects of each type found in a random sample of 100 repositories.

System. This type refers to executable applications designed for providing end-user-oriented functionality. We distinguish systems from libraries (explained shortly), which target developers instead of end-users. ML-enabled systems incorporate a variety of different software assets, both ML-related, such as model files, and non-ML-related, such as configuration files or GUI resources. We also distinguish them from toolsets (explained shortly), which are similar, but focus on specific functionality and are not systems in the sense of being comprised of multiple modules or components integrated with each other. ML-enabled systems are among the largest and most complex projects in our dataset.

Example: An example for a system is "NEWS Headline Classifier Project."[6] It is a web application for classification of topics for news headlines. The repository contains scripts for training the models, binary files of trained models, web-related assets, such as HTML and CSS files, and a Django wrapper.

Library. This type of projects refers to ML libraries, i.e., packaged code to be integrated in other projects—especially ML-enabled systems—via an API. These libraries range from specialized toolkits for ML-related purposes,such as deep learning, computer vision, or natural language processing to more general-purpose libraries that offer a wide array of algorithms and utilities. The libraries in our sample are often well-maintained with active collaborators. They often come with extensive documentation, tutorials, and example projects to assist developers in integrating them into their own work.

Example: A library example is "CTLearn."[7] This library provides functionality for analyzing data from imaging atmospheric Cherenkov telescopes. It contains mostly source files, with functions such as model implementations, but also configuration files. The documentation is mainly provided on an external page. It explains the installation as a Python package and usage of the provided functions.

Toolset. This type refers to ML-related projects that are intended for practical use, but that are very limited in their applicability. In contrast to systems, they lack the property of interconnected components and functionalities. Instead, they consist of a set of isolated functionalities.

Example: An example is the project "Copro."[8] It is a command-line tool for training models to predict conflict risks. As such, it can be executed via shell scripts or by directly interacting with the code.

Summary RQ$_1$: Types of Project

The types of ML-related projects on GitHub are diverse. The majority consist of non-production focused projects, specifically experiments/research projects or tutorials/education projects. In contrast, libraries, toolsets, and actual ML-enabled systems represent minorities. We provided definitions, but the boundaries are not always sharp between the types, especially between education and tutorials, and between toolsets and systems. For the latter, for instance, we relied on a subjective assessment of whether we consider the number of components and their interconnection to be sufficient to call it a system.

[6]https://github.com/AnityaGan9urde/NEWS-Headline-Classifier-project
[7]https://github.com/ctlearn-project/ctlearn
[8]https://github.com/JannisHoch/copro

4.1.2 Discussion. Properly defining the different types of ML-related projects is important to foster future work and build specific methods and tools for the different types of projects. In fact, we have not found such a classification in the literature, so we defined our own types in a bottom-up way. As already discussed, the boundaries are not always clear-cut, as we explained for systems and toolsets. For example, the tool "Copro" we mentioned above was classified as a tool, because the individual functions, like training models and making predictions on a given data, are executed separately from the command line. The project "NEWS headline Classifier" on the other hand was classified as a system, because it is wrapped in the Django framework and offers a web interface. Future work needs to expand on our analysis and definitions, manually analyzing larger samples of projects, and refine our definitions, which is valuable future work, but a study on its own.

The small share of software systems compared to research and education projects is an interesting characteristic of the dataset. It indicates a lack of adoption of ML, which can have many reasons. Whether ML technology is still immature, or whether current open-source software systems are not yet ready to adopt it, is an interesting future research question. However, it definitely confirms that more support is required, especially for developers of small and medium sized open-source projects, possibly through simple and generic tools and frameworks.

Our analysis also revealed that identifying specific types of ML-related projects, such as experiment projects or ML-enabled systems, and distinguishing them (e.g., systems from libraries), is challenging. While it was possible to manually extract small samples of those, it is hard to automatically filter them from a large dataset like the one we present in this paper. Possible solutions could be found by refining the filtering criteria, such as the number of source files or lines of code. Alternatively, classification could be done using ML techniques, such as large language models (LLMs). In fact, building a reliable, perhaps LLM-based classification technique is an open problem. It would constitute valuable future work and enable future large-scale studies of ML-related projects.

4.2 ML Development Stages (RQ$_2$)

Within the full dataset of 31,066 GitHub projects, we were able to identify all six ML development stages (cf. Fig. 6) we are concerned with: `Data Acquisition`, `Data Preparation`, `Modeling`, `Training`, `Evaluation`, and `Prediction`.

4.2.1 Prevalence and Combination of Stages. Among all projects, we found the data acquisition (30,063) and preparation (31,008) stages in over 96 % and 99 %, respectively. The modeling stage occurred fewer times, in 26,808 projects, the training stage in 27,140 projects, the prediction stage in 26,136 projects, and the evaluation stage in 21,894 projects. This shows that most projects involve data acquisition and preparation, while the evaluation stage is the least observed ML stage. We note that the prevalence of these stages is very similar to the results of Biswas et al. [9] obtained on a set of 105 data science projects from Kaggle.

Since the unique combinations of these stages in each project can provide valuable insights into the projects' development, we elicited the frequency of all ML stage combinations. For example, 17,333 projects (over 55 %) implement all of the six ML development stages.

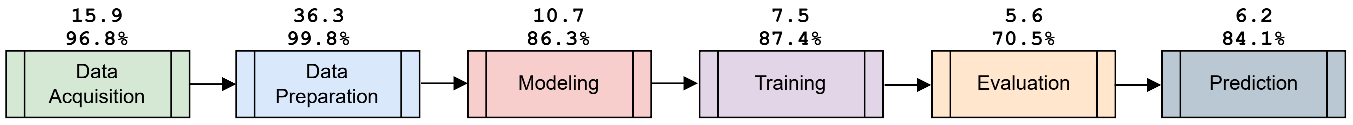

Figure 6: Overview of the individual ML development stages in our subject projects with the average number of associated source files and the share of the projects that contain each stage.

Overall, we identified 51 different combinations of these ML stages, ranging from projects with only one stage to those including all. Figure 7 shows an overview of the most common combinations. Over half of the projects (17,333) implement all ML stages, making this by far the most prominent combination. Only the data preparation stage is present in all combinations, while the evaluation stage is missing most often.

Furthermore, we investigated the implementation of multiple ML stages within the same file. Figure 8 shows the number of stages implemented in each source file on average. This confirms that the majority of files are either associated with no stage (e.g., support files) or can be attributed to a single stage; allowing us to draw conclusions about the evolution of that stage, based on changes to the file. Nevertheless we found some amount of tangling, where multiple stages were implemented in the same file.

We also investigated which stages frequently occur within the same file, as shown in Fig. 9. We found that stages that follow each other in the normal workflow, such as data acquisition and data preparation, often occur together. We also found a substantial number of files where almost all stages were packed together.

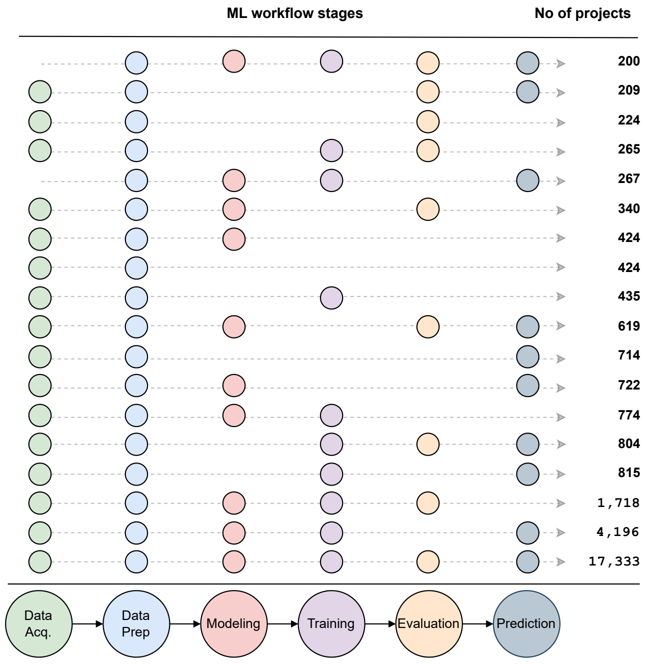

Figure 7: Combination of ML development stages that occur at least 100 times among our subject projects.

Summary RQ$_2$: ML Development Stages

Most ML-related projects cover all typical development stages. Variations in those stages may imply different practices and goals, but the low ratio for evaluations indicates that many projects are exploratory, not focused on training models to be used in practice. While many files seem connected to an individual ML stage, there is tangling that may challenge developers' work and could be facilitated through better traceability and tools.

4.2.2 Discussion. As shown in Fig. 6, the most frequent stage is the `Data Preparation` stage followed by `Data Acquisition`. This indicates that many development activities for ML-enabled projects contribute towards data-oriented stages. Another insight from the stages we found in the projects is the lack of evaluation activities in many of them. This observation is strange and may indicate poor practices, since it implies that those ML projects are not evaluated. However, this may also indicate that many ML projects are exploratory or focus on experimentation without having any intended goal of becoming production-ready. This is in line with our findings from Sec. 4.1, where we found that most projects in our dataset are experiments and tutorials.

More than half of our subject projects involve development activities affecting all observed ML stages. This implies that most ML development activities can be anticipated and expected to follow the established development process sketched in Sec. 2.1. However, while almost all projects include data acquisition and preparation, the modeling, training, and prediction stages occur less often. Some projects also skip modeling altogether, but conduct training and prediction stages. This finding may be a reflection of different ML practices or methods. For instance, projects with prediction stages that skip modeling or training may use pre-trained models.

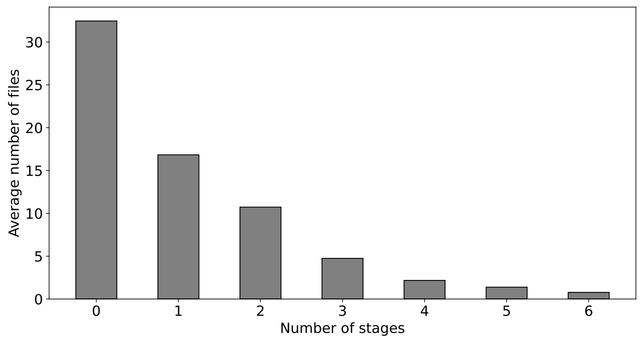

Figure 8: Number of files that are associated with multiple ML stages, average for each project

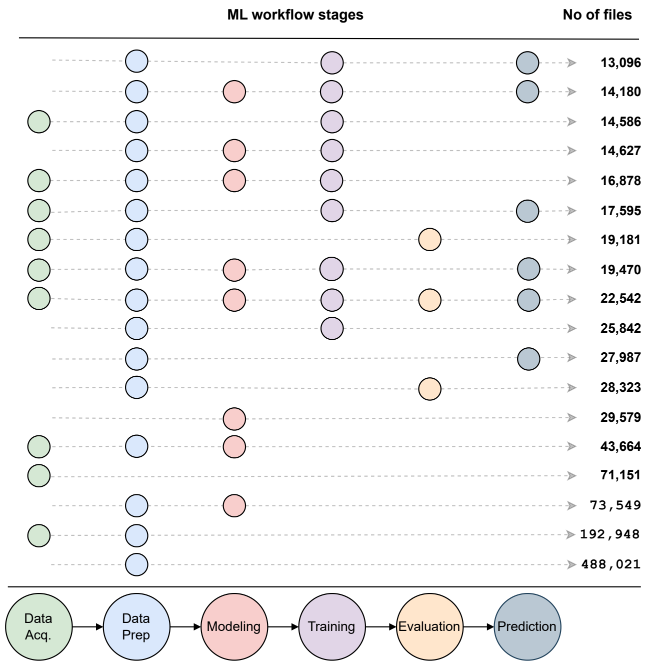

Figure 9: Combination of ML development stages within the same file

The implementation of multiple ML stages within one file occurred more often than expected. Especially the tangling of many stages or stages that are far apart in the workflow raises quality concerns, as it may complicate developers' comprehension of such files. These observations could indicate that some developers are unaware of the established workflow procedures. On the other hand it may also stem from necessities or other requirements in software development that we are unaware of or fail to capture with our empirical approach. Better traceability and tool support for managing such tangled ML development stages and the corresponding assets could potentially help facilitate developers' tasks. The phenomenon is definitely worth further investigation.

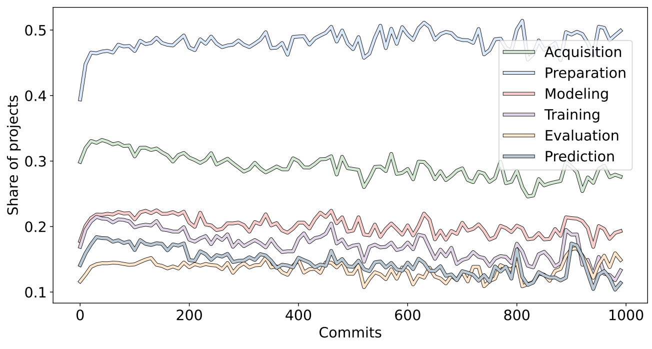

Figure 10: Share of all subject projects changing a specific ML stage at a certain point in the projects development.

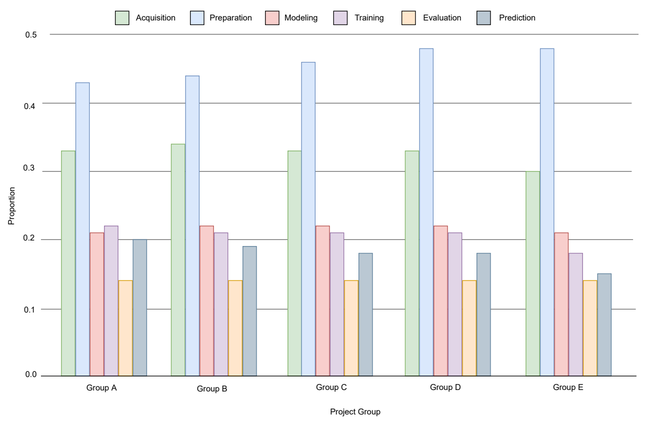

Figure 11: Proportion of project commits affecting each ML stage per group

4.3 ML Project Evolution (RQ$_3$)

For our final research question, we investigated the evolution of ML projects by counting how often the code related to specific ML development stages is modified and how this changes throughout the lifecycle of a project.

4.3.1 Evolution Patterns. Figure 10 presents the proportion of our subjects affecting different ML development stages throughout their lifecycle as observed in their commits. Based on our findings, most project changes predominantly impact the data acquisition and data preparation stages. Approximately 50% of the projects consistently make changes to the stage `Data Preparation` over time. We observed that, aside from the initial few commits, modifications related to data preparation remain consistent throughout the lifecycle of the projects. As for the stage `Data Acquisition`, on average 30% of the projects introduce changes to this stage over time. This activity has a slight downward trend as the number of commits increases.

In contrast, the other stages—Modeling, Training, Evaluation, and Prediction—exhibit relatively stable trends, with only minor decreases over time. The proportion of projects affecting these stages per commit ranges from approximately 5% to 20%

To give a similar but diverse perspective, Fig. 11 presents the proportion of commits affecting each ML stage across groups of projects with different numbers of commits. Data acquisition and preparation are the primary observations of all the project groups (i.e., Group A-E). Similar to the results from Fig. 10, there is no significant change in the pattern observed between shorter and longer projects. Nevertheless, as we progress from projects with fewer commits to those with more commits, the data preparation activities slightly increase, while other activities decrease marginally.

We also measured the average project duration (i.e. the period from the first commit to the last observed commit). Group A has an average duration of 264 days, group B, 357 days, group C, 399 days, group D, 533 days, and group E, 989 days.

Summary RQ$_3$: ML Project Evolution

Data acquisition and preparation remain continuously relevant ML development stages throughout such projects' evolution, requiring corresponding tool support for developers.

4.3.2 Discussion. Building on these results, the data acquisition and preparation stages are the most common and likely expensive activities, requiring adequate tool support for developers. Recall that the statistics in Fig. 4 show that most projects have comparably few commits. So, the changes in evaluation-related stages may be lower due to an abundance of small experimental projects, indicating that developers need tool support for model evaluation.

As the number of commits to a project increases, we would expect to see changes in the stages of development according to the ML workflow described in Sec. 2.1. For example, we would expect the data-related stages to decrease towards the end of a project lifecycle, while the modeling and evaluation stages would increase with more commits. However, our analysis shows that there is no significant shift in the distribution of stages over time. Data acquisition decreases slightly, while data preparation increases slightly with increasing commits, indicating a tendency to have continuous data preparation stages in long-living projects. On the other hand, training and prediction-related stages slightly reduce with an increasing number of commits. Contrary to our assumption, the development stages of affected ML stages seem to remain relatively constant throughout a project's lifespan.

5 IMPLICATIONS

For **RQ$_1$**, we found that ML projects on GitHub span a wide array of different types, with the majority being non-production focused. This indicates a need for further research into how the development of open-source Ml projects can transition from prototypes and experiments to mature software systems.

The small fraction of these projects being actual ML-enabled systems raise the following considerations. A reason could be diverse tool dependencies (i): One possible explanation for the limited presence is the complex nature of such system, which mostly require developers to rely on an array of specialized tools serving a unique purpose. It is plausible that Git repositories are used to manage a subset of all assets, such as documentation and code, while other assets, such as data, trained models, and so on, are managed through alternative tools. Another reason could be the isolation of ML experiments and software systems (ii): Our findings could also be interpreted to support the theory that ML experiments and software system development are often conducted in isolation rather than as an integrated project. This separation may arise from the historical division between data science and software engineering teams, resulting in disjoint workflows and project structures. For example, there could be separate teams or developers working on different aspects of ML-based systems. For instance, the ML component could be managed as a Git project by a team while a separate team develops the software systems utilizing the model.

These findings illustrate a need for novel studies that specifically identify ML-enabled systems on a large scale. This, however, requires designing automated project type identification techniques (cf. Sec. 4.1.2) that can be used to drive future studies. In addition, identifying the reasons behind the limited presence of systems requires exploratory studies, relying on interviews or surveys with developers, also confirming or refuting our results.

Our results for **RQ$_2$** and **RQ$_3$** show that the most important aspects in the development of ML-enabled software are related to data acquisition and preparation. This finding confirms that data management is typically identified as the biggest challenge when building software with ML models, such as ML-enabled systems [24–26]. These findings demand improved tool support for DataOps, specifically for making changes to the data-related parts of ML pipelines.

Also recall that including ML requires many different assets (e.g., datasets, hyperparameters). In combination with our results and the highly iterative nature of ML development, novel tool support for managing the evolution is needed. Current version-control systems as known from software engineering (e.g., Git) are insufficient, while specialized tools, such as experiment management tools, are not well integrated with existing software-engineering tooling, especially version-control systems. Since our results indicate that ML models and software (sub-)systems seem to be developed in isolation, we recommend the development of tools that facilitate the unified management of ML experiment assets with version-controlled repositories. These tools should seamlessly handle code, datasets, models, and experimentation records, making Git a more suitable platform for ML projects targeting software systems.

6 THREATS TO VALIDITY

We now discuss possible threats to the validity of our study.

6.1 External Validity

To enhance external validity, we focused on ML-related projects developed with Python, the primary language used in developing ML components. However, other programming languages should also be examined to further investigate and confirm or refute our results and observations. Furthermore, we filtered projects using only two ML libraries and their APIs. However, TensorFlow and scikit-learn are among the most popular ML libraries, and we see our study as a comprehensive starting point for further investigations in this direction. The other filtering criteria we employed, such as the number of commits or the presence of ML workflow stages were chosen heuristically and are therefore subject to possible biases. We leave it to future research to evaluate their effectiveness and propose possible alternatives. Finally, recall that there are multiple possible workflows for ML development, comprising different stages. We focused on the six stages that were already defined and analyzed by Biswas et al. [9], because they could be found in GitHub repositories. Future work could build on our study by also taking other stages into account, which would require a different methodology and a separate study.

6.2 Internal Validity

We note that our dataset includes a diverse array of project types (cf. Sec. 4.1). The analysis performed in sections 4.2 and 4.3 did not capture how the implementation and usage of ML development stages differs between project types. For example, it would be possible that the lack of evaluation activities is caused by one or two project types, such as libraries or tutorials. Differentiating these categories and analyzing them individually is beyond the scope of this paper and would be a study of its own. Furthermore the counted number of projects does not take into account how large the projects are. It might be possible that some categories, such as systems or libraries contain projects that are larger than the projects in other categories. We also note that the manual classification of project types is subject to possible errors and biases by the authors, who might have a different understanding of these categories than someone else.

Finally, to design and implement our mining methodology, we have considered the limitations and challenges of mining repositories raised by Cosentino et al. [11] and Gousios et al. [15] to avoid common pitfalls, including low-level replicability, poor sampling techniques, inefficient collection process, and biased results.

7 CONCLUSION

We presented a large-scale study on ML-related Python projects on GitHub. We studied what types of projects are ML-enabled, the occurrence of development stages, and the dominant evolution activities found in over 31,000 repositories on GitHub. Our work contributes to a growing research interest in improving the practices and engineering of ML-enabled software projects. We find that most ML-related projects are research- and education-oriented, indicating that the technology is not yet commonly integrated in major open-source software systems. Furthermore, our findings show that data-related development stages dominate these projects, whose related assets are also most frequently changed. Notably, we found the model evaluation stage in the least number of projects, followed by the training and prediction stages. Finally, we observed that, while the distribution of ML stages is consistent as projects evolve with increasing commits, the proportion of data preparation increases and data acquisition decreases. The outcome of our study offers insights into essential aspects of ML development stages and asset types that tool developers and researchers should prioritize.

ACKNOWLEDGMENTS

Supported by Berger's fellowship granted by the Royal Swedish Academy of Sciences and the Wallenberg Foundation.

REFERENCES

[1] 2021. Most popular machine learning libraries - 2014/2021. https://statisticsanddata.org/data/most-popular-machine-learning-libraries
[2] Mohannad Alahdab and Gül Çalıklı. 2019. Empirical analysis of hidden technical debt patterns in machine learning software. In *PROFES*.
[3] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *ICSE/SEIP*.
[4] A Arpteg, B Brinne, L Crnkovic-Friis, and J Bosch. 2018. Software Engineering Challenges of Deep Learning. In *SEAA*.
[5] Amine Barrak, Ellis E Eghan, and Bram Adams. 2021. On the co-evolution of ml pipelines and source code-empirical study of dvc projects. In *SANER*.
[6] Andrew L Beam, Arjun K Manrai, and Marzyeh Ghassemi. 2020. Challenges to the reproducibility of machine learning models in health care. *Jama* 323, 4 (2020), 305–306.
[7] Aaditya Bhatia, Ellis E Eghan, Manel Grichi, William G Cavanagh, Zhen Ming, Bram Adams, et al. 2022. Towards a Change Taxonomy for Machine Learning Systems. *arXiv preprint arXiv:2203.11365* (2022).
[8] Sumon Biswas, Md Johirul Islam, Yijia Huang, and Hridesh Rajan. 2019. Boa meets python: a boa dataset of data science software in python language. In *MSR*.
[9] Sumon Biswas, Mohammad Wardat, and Hridesh Rajan. 2021. The Art and Practice of Data Science Pipelines: A Comprehensive Study of Data Science Pipelines In Theory, In-The-Small, and In-The-Large. *arXiv:2112.01590* (2021).
[10] Dan Bohus, Sean Andrist, and Mihai Jalobeanu. 2017. Rapid development of multimodal interactive systems: a demonstration of platform for situated intelligence. In *ICMI*.
[11] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2016. Findings from GitHub: methods, datasets and limitations. In *MSR*.
[12] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Commun. ACM* 39, 11 (1996), 27–34.
[13] Rudolf Ferenc, Tamás Viszkok, Tamás Aladics, Judit Jász, and Péter Hegedüs. 2020. Deep-water framework: The Swiss army knife of humans working with machine learning models. *SoftwareX* 12 (2020), 100551.
[14] Danielle Gonzalez, Thomas Zimmermann, and Nachiappan Nagappan. 2020. The state of the ml-universe: 10 years of artificial intelligence & machine learning software development on github. In *MSR*.
[15] Georgios Gousios and Diomidis Spinellis. 2017. Mining software engineering data from GitHub. In *ICSE-C*.
[16] Samuel Idowu, Osman Osman, Daniel Strueber, and Thorsten Berger. 2022. On the Effectiveness of Machine Learning Experiment Management Tools. In *44th International Conference on Software Engineering, Software Engineering in Practice track (ICSE/SEIP)*.
[17] Samuel Idowu, Daniel Strüber, and Thorsten Berger. 2022. Asset Management in Machine Learning: State-of-research and State-of-practice. *Comput. Surveys* 55, 7, Article 144 (dec 2022), 35 pages.
[18] Samuel Idowu, Daniel Strueber, and Thorsten Berger. 2022. EMMM: A Unified Meta-Model for Tracking Machine Learning Experiments. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*.
[19] Samuel Idowu, Daniel Strüber, and Thorsten Berger. 2021. Asset Management in Machine Learning: A Survey. In *ICSE/SEIP*.
[20] Richard Isdahl and Odd Erik Gundersen. 2019. Out-of-the-Box Reproducibility: A Survey of Machine Learning Platforms. In *eScience*.
[21] Microsoft. 2017. Team Data Science Process Documentation. https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/
[22] Ml-Tooling. [n. d.]. ML-tooling/best-of-ml-python: A ranked list of awesome machine learning python libraries. updated weekly. https://github.com/ml-tooling/best-of-ml-python
[23] Marçal Mora-Cantallops, Salvador Sánchez-Alonso, Elena García-Barriocanal, and Miguel-Angel Sicilia. 2021. Traceability for trustworthy AI: A review of models and tools. *Big Data and Cognitive Computing* 5, 2 (2021), 20.
[24] Aiswarya Raj Munappy, Jan Bosch, and Helena Homström Olsson. 2020. Data pipeline management in practice: Challenges and opportunities. In *PROFES*.
[25] Aiswarya Raj Munappy, Jan Bosch, Helena Holmström Olsson, Anders Arpteg, and Björn Brinne. 2022. Data management for production quality deep learning models: Challenges and solutions. *Journal of Systems and Software* 191 (2022), 111359.
[26] Aiswarya Raj Munappy, David Issa Mattos, Jan Bosch, Helena Holmström Olsson, and Anas Dakkak. 2020. From ad-hoc data analytics to dataops. In *ICSSP*.
[27] Nadia Nahar, Haoran Zhang, Grace Lewis, Shurui Zhou, and Christian Kästner. 2023. A Dataset and Analysis of Open-Source Machine Learning Products. arXiv:2308.04328 [cs.SE]
[28] Luigi Quaranta, Fabio Calefato, and Filippo Lanubile. 2021. A taxonomy of tools for reproducible machine learning experiments. *AIxIA* (2021).
[29] Dhivyabharathi Ramasamy, Cristina Sarasua, Alberto Bacchelli, and Abraham Bernstein. 2023. Workflow analysis of data science code in public GitHub repositories. *Empirical Software Engineering* 28, 1 (2023), 1–47.
[30] Sebastian Raschka and Vahid Mirjalili. 2019. *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd.
[31] Iqbal H Sarker, Faisal Faruque, Ujjal Hossen, and Atikur Rahman. 2015. A Survey of Software Development Process Models in Software Engineering. *International Journal of Software Engineering and Its Applications* 9, 11 (2015), 55–70.
[32] Marius Schlegel and Kai-Uwe Sattler. 2022. Management of Machine Learning Lifecycle Artifacts: A Survey. *SIGMOD Rec.* (2022), 18–35.
[33] Andrew J Simmons, Scott Barnett, Jessica Rivera-Villicana, Akshat Bajaj, and Rajesh Vasa. 2020. A large-scale comparative analysis of coding standard conformance in open-source data science projects. In *ESEM*.
[34] Rachael Tatman, Jake Vanderplas, and Sohier Dane. 2018. A Practical Taxonomy of Reproducibility for Machine Learning Research. In *ICML*.
[35] Bart van Oort, Luís Cruz, Maurício Aniche, and Arie van Deursen. 2021. The Prevalence of Code Smells in Machine Learning projects. In *WAIN*.
[36] Jiawei Wang, Li Li, and Andreas Zeller. 2021. Restoring execution environments of Jupyter notebooks. In *ICSE*.
[37] Thomas Weißgerber and Michael Granitzer. 2019. Mapping platforms into a new open science model for machine learning. *it - Information Technology* 61, 4 (2019), 197–208.
[38] Rüdiger Wirth and Jochen Hipp. 2000. CRISP-DM : Towards a standard process model for data mining. In *KDD*.
[39] Yue Yu, Gang Yin, Huaimin Wang, and Tao Wang. 2014. Exploring the patterns of social behavior in GitHub. In *CrowdSoft*.
[40] Amy X Zhang, Michael Muller, and Dakuo Wang. 2020. How do data science workers collaborate? roles, workflows, and tools. *Proc. of the ACM on Human-Computer Interaction* 4, CSCW1 (2020), 1–23.