DVC in Open Source ML-development: The Action and the Reaction

Lorena Barreto Simedo Pacheco
Concordia University
Montréal, Canada
lorena.bspacheco@mail.concordia.ca

Musfiqur Rahman
Concordia University
Montréal, Canada
musfiqur.rahman@mail.concordia.ca

Fazle Rabbi
Concordia University
Montréal, Canada
fazle.rabbi@mail.concordia.ca

Pouya Fathollahzadeh
Queen's University
Kingston, Canada
pouya.fathollahzadeh@queensu.ca

Ahmad Abdellatif
University of Calgary
Calgary, Canada
ahmad.abdellatif@ucalgary.ca

Emad Shihab
Concordia University
Montréal, Canada
emad.shihab@concordia.ca

Tse-Hsun (Peter) Chen
Concordia University
Montréal, Canada
tse-hsun.chen@concordia.ca

Jinqiu Yang
Concordia University
Montréal, Canada
jinqiu.yang@concordia.ca

Ying Zou
Queen's University
Kingston, Canada
ying.zou@queensu.ca

ABSTRACT

Machine Learning (ML) systems are gaining popularity, reshaping various domains ranging from customer services to software engineering. The effectiveness of ML systems is dependent on the quality of their training data. Therefore, practitioners invest substantial time experimenting with different data, parameters, and models to guarantee the quality of the end system. Prior work highlighted unique challenges of developing ML systems, particularly concerning versioning data and models. Recently, various tools such as DVC and MLFlow have emerged to aid developers in the storage and tracking of data. Despite their growing popularity, very little is known about their usage patterns and impact on open-source software (OSS) systems. To address this gap, we conducted an empirical study on 56 GitHub OSS projects that use DVC to understand the DVC usage pattern and the impact of using DVC on the software development process. We found that *Versioning and tracking* is the most adopted DVC feature, being utilized by all 56 projects and being the only adopted feature in 85.7% of them. Furthermore, we found that DVC has a significant impact on the software development process indicators such as the number of created pull requests (PRs), and the number of bug-fix commits. For instance, our findings showed that DVC causes a peak in the number of commits and PRs at the moment of the adoption, followed by a long-term decrease. We believe that our findings can assist practitioners in tailoring tools to better meet user requirements and help organizations realize potential outcomes of adopting such tools.

CAIN 2024, April 2024, Lisbon, Portugal

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

CCS CONCEPTS

• **Software and its engineering → Software configuration management and version control systems**.

KEYWORDS

Empirical Software Engineering, Data Version Control, Software Evolution, SE4AI

ACM Reference Format:
Lorena Barreto Simedo Pacheco, Musfiqur Rahman, Fazle Rabbi, Pouya Fathollahzadeh, Ahmad Abdellatif, Emad Shihab, Tse-Hsun (Peter) Chen, Jinqiu Yang, and Ying Zou. 2024. DVC in Open Source ML-development: The Action and the Reaction. In *Proceedings of 3rd International Conference on AI Engineering — Software Engineering for AI (CAIN 2024).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Machine Learning (ML) systems have diverse applications spanning different domains, such as the detection of spam emails [11], chatbots [10], and AI-assisted surgery [23] to name a few. Models and data constitute the backbone of ML-based systems. Previous research demonstrates that training models often involve a substantial amount of experimentation with different types of data and modeling techniques to achieve the best-performing models [16]. Consequently, version control for data, models, and pipeline configurations is essential to maintain transparency and reproducibility throughout the development and deployment of ML systems. Recently, numerous tools (*e.g.,* DVC and MLFlow) have emerged, empowering ML developers to efficiently track, version, and build ML components [19]. DVC [1] is an open-source tool for versioning of ML artifacts, pipeline building, and experiment tracking. Due to its easy-to-use nature, it is gaining traction among practitioners. Prior work shows a significant adoption trend of different DVC features in the open-source software (OSS) community [8]. However, the usage patterns and impact of using DVC remain open questions. Addressing these aspects is crucial for advancing the understanding of the integration of version control tools in the dynamic landscape

of ML application development, while also underscoring the advantages of incorporating DVC within organizations.

In this paper, we conduct an empirical study using 56 GitHub projects to investigate how DVC is used in OSS development. More specifically, our study answers the following research questions:

RQ1: (The action) How is DVC used in open-source ML-based software development?

We find that developers use DVC's *Versioning and tracking* feature more frequently than DVC's *Pipelining* feature. Specifically, 85.7% of projects use DVC exclusively for *Versioning and tracking*, while all the remaining projects use DVC for both *Versioning and tracking* as well as *Pipelining*.

While RQ1 focuses on the adoption and usage patterns of different DVC features, in RQ2, we aim to understand the impact of adopting DVC in the development process.

RQ2: (The reaction) How does the introduction of DVC affect the software development process?

We analyze 6 software development process indicators and find that the majority of indicators show a shift in existing trends after the introduction of DVC. For example, with the DVC adoption, the number of created pull requests (PRs) decreases.

In this work, our contributions are as follows:

(1) To the best of our knowledge, this is the first work that studies the DVC usage patterns and the impact of DVC adoption on the software development process.
(2) We make our code and dataset publicly available [1] to accelerate future research in this area.

Paper Organization. The rest of the paper is organized as follows. In Section 2, we provide an overview of the study's background, while in Section 3, we discuss the dataset used in our analysis. Section 4 presents our findings for each RQ. In Sections 5 and 6, we cover related works and potential threats to validity, respectively. Finally, we conclude the paper by outlining directions for future research in Section 7.

2 BACKGROUND

DVC is an open-source tool designed for data and model versioning. DVC enables practitioners to version and track their ML artifacts, reproduce experiments, and monitor their ML pipelines. DVC operates on top of Git repositories and cloud storage. DVC comprises two main features, namely, *Versioning and tracking* feature and *Pipelining* feature. In this section, we explain each feature and the associated terminologies used throughout the paper.

2.1 Versioning and tracking

DVC can be used to version and track different ML components of a system. The *Versioning and tracking* functionalities allow practitioners to store and track large datasets and ML models on the cloud without having to keep them in the code repository, which increases the security and privacy of ML artifacts and reduces their coupling with general software artifacts. The `.dvc` file keeps track of all changes that occur to the data and/or models. Listing 1 presents a snapshot of one of the `.dvc` files from the project `dna-seq` [3]. Line 1 represents the MD5 hash for the `.dvc` file itself. `deps` (line 2) and `outs` (line 6) represent the dependency and output blocks, respectively. Lines 3-5 represent the dependency entries, `path` in line 3 is the path to the dependency, `size` in line 4 represents the size of the dependency file/folder, and `etag` in line 5 is the ETag for HTTP [5]. The `md5` in line 7 represents the MD5 hash value for the file/directory being tracked with DVC. As shown in line 3, the file being tracked by DVC is saved remotely and tracked via its URL, which makes the usability of DVC flexible.

```
1  md5: 531ba86b382cbb71a7ed0fad1be2ccd1
2  deps:
3  - path: http://ftp.ncbi.nlm.nih.gov/pub/clinvar/
       vcf_GRCh38/clinvar.vcf.gz
4    size: 57813301
5    etag: '"3722935-5df53b38168c8"'
6  outs:
7  - md5: 1a0bd5729abb411a388e590431c62f99
```

Listing 1: Example of a `.dvc` file.

2.2 DVC for ML Pipelines

In addition to its *versioning* feature, DVC enables workflow automation via its *pipelining* feature. This attribute allows teams to use DVC to define ML pipelines to perform tasks like data collection, model training as well as validation. Each DVC pipeline is represented by a `dvc.yaml` file. For example, Listing 2 shows a `dvc.yaml` file from the project `guildai` [4]. The DVC pipeline is composed of an aggregate of stages. Each stage represents a process that typically receives an input produced in the previous stage and delivers an output file. In this example, the partial contents from the original file are shown. The `prepare-data` stage, among other stages, is shown in line 2. In the `prepare-data` stage, the pipeline executes a Python script (line 3) which performs data preprocessing and prepares the data to be used in modelling. The `deps` and `outs` block define the dependencies and the produced output files, respectively.

```
1  stages:
2    prepare-data:
3      cmd: python prepare_data.py
4      deps:
5      - iris.csv
6      - prepare_data.py
7      outs:
8      - iris.npy
9    ...
```

Listing 2: Example of a `dvc.yaml` file.

It is worth noting that both DVC functionalities are independent of each other. For example, practitioners can use DVC only to version ML artifacts while using a different tool to define ML pipelines.

3 DATASET

The main goal of this study is to investigate DVC usage and the impact of its adoption in OSS projects. Therefore, we collect projects from GitHub that use DVC. In this section, we detail the process of curating our dataset to answer the RQs.

Software projects that use DVC have the following artifacts: `*.dvc`, `dvc.lock`, `dvc.ignore` and `dvc.yaml`. We leverage GitHub's API search to identify projects containing one or more files with the term "dvc" in the file name. This step returns 168 projects. Given that the goal of this study is to understand the usage patterns of

[1] https://zenodo.org/records/10161505

Table 1: Descriptive Statistics of the Dataset

Metrics	Avg.	Std.	Min.	Median	Max.
Stars	142.84	508.66	0	18	3673
Commits	1410.35	3770.12	10	287	25931
DVC Commits [a]	46.23	87.78	0	18	557
Contributors	7.46	9.17	1	4	30

[a] Any commit that modifies one or more DVC artifacts is considered a DVC commit.

DVC within open-source development, we set out to collect non-toy projects that use DVC. To filter out toy projects from the returned results, two authors independently annotate each project by analyzing each project's name, its description, and the README file. This process filters out any project where DVC is used for educational, tutorial, or experimental purposes. The annotators achieve a substantial agreement ($kappa = 0.96$) while classifying 168 projects returned from the GitHub query. In cases of disagreement, all the authors meet to revisit the projects and discuss them until they reach an agreement. The annotation process results in 56 real projects. Table 1 presents the statistics of the collected non-toy projects. From the table, we observe that our dataset contains projects that vary in terms of size and popularity.

To assess DVC's adoption impact, it is necessary to have a reasonable amount of data for the same duration of time before and after the tool's introduction [15]. Similar to prior works [9, 12], we choose a period of 26 weeks or approximately 6 months in our analysis. Therefore, in RQ2, we use a subset of the aforementioned 56 projects in which data is available for 26 weeks before and 26 weeks after DVC adoption. A total of 16 projects meet this criterion.

4 RESULTS AND FINDINGS

In this section, we describe our findings for each research question defined in Section 1.

4.1 RQ1: (The action) How is DVC used in open-source ML-based software development?

Motivation: Prior work shows an early adoption and high usage of DVC functionalities (*versioning* and *pipelining*) in more than a quarter of their studied projects [8]. However, exploring the DVC features that are being used by the ML-developers remains an open question. Understanding the usage patterns of DVC provides insights into the specific DVC features and workflows employed by ML developers. Therefore, in this RQ, we set out to identify how ML developers are using DVC in their software projects.

Approach: To gain insights into how the OSS community uses DVC, we categorize the projects in our dataset into the following categories:

- ***Versioning and tracking*:** Projects that use the *Versioning and tracking* feature exclusively.
- ***Pipelining*:** Projects that use the *Pipelining* feature only.
- **Both:** Projects that use both features.

We employ a set of heuristics to classify projects based on their DVC usage. Specifically, a project containing files with the `.dvc` extension indicates that the project uses DVC for ***Versioning and tracking***, as discussed in Section 2. On the other hand, if a project contains one or more `dvc.yaml` files, it indicates the use of DVC for constructing ML pipelines. In such cases, we classify the project as ***Pipelining***. We categorize a project as **Both** if it incorporates both `.dvc` and `dvc.yaml` files.

Results: We find that 85.7% of the projects use DVC only for versioning artifacts like data files. Since data plays a pivotal role in ML applications and the core functionality of DVC lies in data versioning, DVC is a suitable choice to track the data utilized in training and testing ML models. Additionally, the result reveals that 14.3% of the projects employ DVC for both data versioning and building ML pipelines, however, none of the projects use DVC exclusively for constructing ML pipelines. This might be due to the preference of practitioners who opt for a different tool like MLFlow [6]) for their ML pipeline development. To better understand the ML pipeline stages that practitioners create using DVC, we manually examine the 29 `dvc.yaml` files. The first two authors independently inspect the stage names and code for each DVC pipeline. They categorize each stage in the pipeline using the categories described in [7, 24]: exploratory data analysis (EDA), data collection, data pre-processing, data cleaning, model training, model validation, model fine-tuning, and post-processing. The categorization process achieves a complete agreement ($kappa = 1$) between the authors.

We find that among all the projects that use DVC for building pipelines, 75% of the projects use DVC for data preprocessing and cleaning, 63% define model training, and 38% define model validation as part of the pipeline. Although these 3 stages (data preprocessing, model training, and model validation) are widely considered to be the core ML-related stages, there are other interesting use cases of the pipeline-building feature as well. For example, we find that practitioners use DVC pipeline feature to define non-core ML stages such as data fetching from the cloud, decompressing compressed files, and spinning up containers in the pipelines.

4.2 RQ2: (The reaction) How does the introduction of DVC affect the software development process?

Motivation: Previous research demonstrates that introducing a new tool can be risky due to the inherent uncertainty in software systems [21]. Understanding the impact of adopting DVC on software development is crucial for practitioners to make informed decisions regarding the potential pros and cons of such adoptions. Therefore, in this RQ, we examine the impact of integrating DVC on a set of software development process indicators.

Approach: To investigate the impact of DVC adoption on a software project, we perform an Interrupted Time-Series (ITS) analysis [22]. ITS is a quasi-experimental research design used to evaluate the effect of an intervention (such as the DVC adoption) on an indicator (such as the number of commits) over time as shown in prior studies [15, 20, 28]. ITS compares periods before and after the intervention, assuming the trend would hold if the intervention had not happened. We measure the impact of DVC adoption on 6 software development process indicators [2] as shown in Table 2.

[2]We use 'indicator(s)' in the rest of the paper to shorten the term 'software development process indicator(s)'.

Table 2: Overview of the indicators studied to analyze the impact of DVC adoption.

Indicator	Rationale
# of commits	We analyze the number of commits, number of created PRs and number of closed PRs to understand the impact of DVC on the activity level of a project [25, 26].
# of created PRs	
# of closed PRs	
# of bug-fix commits	We analyze the total number of bug-fix commits to understand the complexity associated with adopting a new tool. A higher number of bug-fix commits in DVC artifacts indicates a higher level of complexity. We use keywords (such as, 'bug', 'fix', 'wrong', 'error', 'fail', 'problem' and 'patch') used in prior work to identify the bug-fix commits [13, 14].
# of contributors	We analyze the number of contributors [15] and number of new contributors to understand DVC's impact on the size and involvement of the project community.
# of new contributors	

To perform ITS analysis in this study, we employ the following linear regression model:

$$Y_t = \beta_0 + \beta_1 \times time_t + \beta_2 \times adoption_t + \beta_3 \times time\_since\_adoption_t + \beta_4 \times controls \quad (1)$$

The variables in Equation 1 are defined as follows:

- Y_t is the dependent variable that represents an indicator (*e.g.,* number of commits and number of contributors) at time t.
- $time$ represents the number of weeks after the start of the observation period at t. $time$ starts at 1 and increments by 1 for each subsequent week.
- $adoption$ represents whether or not DVC has been adopted at t. The value of this variable is 0 before the DVC adoption and 1 after it.
- $time\_since\_adoption$ represents the number of weeks passed since the adoption of DVC at t. The value of this variable is 0 before the DVC adoption, starts at 1 at the time of DVC adoption, and increments by 1.
- $controls$ is calculated via an arithmetic mean of three control variables: the age of the project at the adoption time in weeks to capture the level of maturity of the project, the total number of contributors to capture the size of the project community, and the total number of PRs to capture the level of activity in the project.

To evaluate the fit of the models, we use the marginal R^2 and conditional R^2 for mixed-effects models [17].

Results: The results obtained from each model for all 6 indicators are shown in Table 3. From the table, we observe that all indicators (except the number of new contributors) have a statistically significant (confidence level of 95%) association with *time_since_adoption*, which implies that the trends change after the adoption of DVC.

Figure 1 represents how the adoption of DVC impacts the indicators' long-term trends. The first three indicators, namely *Number of commits*, *Number of created PRs* and *Number of closed PRs*, show a very similar trend of a slight increase at the DVC adoption time followed by a consistent drop afterward. For example, in Figure 1b, we observe that there is an increase in the number of PRs at the adoption time. Then, the number of PRs decreases after the DVC's adoption. This reduction can be attributed to the separation of data from the codebase. In other words, data scientists and developers can independently work on data, reducing the necessity for PRs related to data changes.

Figure 1d shows the change in trend for *Number of bug-fix commits*. Interestingly, we find that there is a sudden spike in bug-fix commits at the time of adoption, although DVC usage leads to a sustained decrease in bug-fix commits in the long run after the tool's adoption. Upon qualitative analysis, we find that the majority of these bug-fix commits are not associated with DVC artifacts. One possible reason for this observation is that practitioners fix the bugs in their code base to prepare the repository for integration with the DVC tool. One example is the DVC introducing commit [3] at the `Search` [2] project. This commit fixes bugs related to directory name updates, changing the `README` file, and updating the library version. Although we cannot say that all these fixes are forced by DVC adoption, many of these fixes are happening concurrently with DVC's incorporation, which supports the finding from [8] regarding the co-evolution of DVC and general software artifacts.

5 RELATED WORK

Recently, many researchers studied the versioning of ML data and models [8, 18, 20, 27]. For example, Njomou *et al.* [18] analyzed the challenges related to model evolution and data versioning for ML software. They found DVC to be a powerful tool and proposed a framework to preprocess the software repository to integrate with DVC. Also, Barrak *et al.* [8] empirically studied the usage of DVC in GitHub projects and demonstrated that ML versioning is a developing trend in open-source repositories. Their finding indicated a high coupling between DVC artifacts and general software artifacts. Schlegel *et al.* [20] provided an overview of ML artifact management systems while Zaharia *et al.* [27] proposed MLflow–a tool designed for building ML pipeline and managing ML projects–with a focus on ease of experimentation, reproducibility, and model deployment.

While our work shares a common goal with the studies mentioned above, which is to investigate data and model versioning tools, to the best of our knowledge, no prior work has explored the usage patterns of DVC or analyzed the impact of adopting DVC in OSS, which is the primary objective of this work.

6 THREATS TO VALIDITY

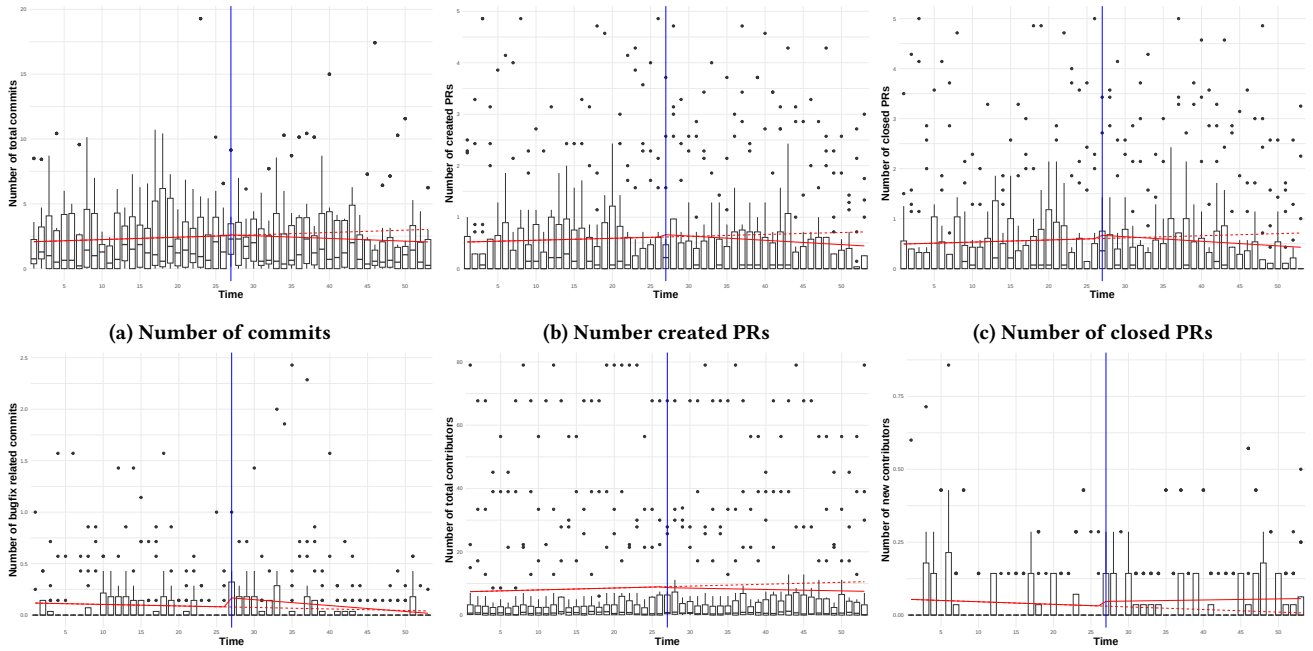
DVC is a new addition to the toolbox of ML developers. Although, the OSS community is adopting this tool at a fast rate [8], to date, majority of the OSS projects hosted on GitHub that are using DVC

[3] commit id: `b06c80cd3192a067438521af2f511d4014a71cd9`

Table 3: Coefficients obtained from each model for all 6 indicators.

	intercept	*time*	*adoption*	*time_since_adoption*	*controls*	Performance in R^2	
						Marginal	Conditional
# of commits	1.654922	0.018221	0.092663	-0.039216 *	0.001253	0.05	0.68
# of created PRs	7.479e-02	3.759e-03	5.598e-02	-1.213e-02 *	1.363e-03 *	0.59	0.71
# of closed PRs	5.401e-02	4.236e-03	7.738e-02	-1.342e-02 *	1.347e-03 *	0.57	0.68
# of bug-fix commits	1.38e-01 *	-1.467e-03	9.261e-02 *	-4.263e-03 *	-6.11e-05	0.03	0.31
# of contributors	0.189602	0.060345 *	-0.10540	-0.104426 *	0.0222	0.5	0.96
# of new contributors	4.542e-02 *	-8.883e-04	1.512e-02	1.258e-03	2.863e-05	0.02	0.36

* The results are statistically significant.



(a) Number of commits **(b) Number created PRs** **(c) Number of closed PRs**

(d) Number of bug-fix commits **(e) Number of contributors** **(f) Number of new contributors**

Figure 1: Interrupted Time-series Analysis plots, in which the blue line defines the DVC adoption time, the solid red line shows the indicators' real values and the dashed red line represents the trend in the indicator had DVC not been adopted.

for purposes like experimentation, tutorials, and demonstration of the tool. Therefore, from our search on GitHub, we end up with a set of 56 non-toy projects, which is a relatively small dataset. We acknowledge that it may not be ideal to generalize our findings across projects of different size, level of activity and age. However, the primary aim of this study is to report the current adoption trend of DVC features, and the current evolution trend of projects after DVC is adopted.

7 CONCLUSION AND FUTURE WORK

In this research, we conduct an empirical study to understand how DVC is used in the OSS community and the impact of using DVC on the software development process. To achieve this, we analyze 56 GitHub OSS projects that use DVC. We find that DVC is mainly used (85.7% of the studied projects) for its *Versioning and tracking* feature, and no project in our dataset adopts DVC for the sole purpose of ML *Pipelinging*. Furthermore, we find that the DVC adoption significantly shifts the existing trends in all but one software development process indicator implying its substantial impact on the overall development process. Our study sheds light on the prevalent usage patterns and impact of DVC in the context of ML-based systems. Moreover, it guides future investigations into best practices for incorporating DVC in open-source projects.

As potential future works, we plan to explore the versioning feature in DVC at a granular level, distinguishing between versioning for data and models. Furthermore, we will conduct surveys among practitioners to gain a better understanding of the trends that emerge after adopting DVC and assess the impact of this shift on the project's progress.

REFERENCES

[1] Data version control · dvc. https://dvc.org/. (Accessed on 11/20/2023).

[2] Github - bluebrain/search: Blue brain text mining toolbox for semantic search and structured information extraction. https://github.com/BlueBrain/Search. (Accessed on 11/20/2023).

[3] Github - dna-seq/dna-seq: Dna-seq pipeline. https://github.com/dna-seq/dna-seq. (Accessed on 11/20/2023).

[4] Github - guildai/guildai: Experiment tracking, ml developer tools. https://github.com/guildai/guildai. (Accessed on 11/20/2023).

[5] Http etag - wikipedia. https://en.wikipedia.org/wiki/HTTP_ETag#Strong_and_weak_validation. Accessed: 2022-10-24.

[6] Mlflow - a platform for the machine learning lifecycle | mlflow. https://mlflow.org./. (Accessed on 11/20/2023).

[7] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.

[8] A. Barrak, E. E. Eghan, and B. Adams. On the co-evolution of ml pipelines and source code-empirical study of dvc projects. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 422–433. IEEE, 2021.

[9] E. Bouwers, A. van Deursen, and J. Visser. Evaluating usefulness of software metrics: an industrial experience report. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 921–930. IEEE, 2013.

[10] J. Cahn. Chatbot: Architecture, design, & development. *University of Pennsylvania School of Engineering and Applied Science Department of Computer and Information Science*, 2017.

[11] P. Garg and N. Girdhar. A systematic review on spam filtering techniques based on natural language processing framework. In *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 30–35. IEEE, 2021.

[12] T. Hall and N. Fenton. Implementing effective software metrics programs. *IEEE software*, 14(2):55–65, 1997.

[13] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella. Taxonomy of real faults in deep learning systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1110–1121, 2020.

[14] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan. A comprehensive study on deep learning bug characteristics. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 510–520, 2019.

[15] S. Khatoonabadi, D. E. Costa, S. Mujahid, and E. Shihab. Understanding the helpfulness of stale bot for pull-based development: An empirical study of 20 large open-source projects. *arXiv preprint arXiv:2305.18150*, 2023.

[16] G. Lorenzoni, P. Alencar, N. Nascimento, and D. Cowan. Machine learning model development from a software engineering perspective: A systematic literature review. *arXiv preprint arXiv:2102.07574*, 2021.

[17] S. Nakagawa and H. Schielzeth. A general and simple method for obtaining r2 from generalized linear mixed-effects models. *Methods in ecology and evolution*, 4(2):133–142, 2013.

[18] A. T. Njomou, A. J. B. Africa, B. Adams, and M. Fokaefs. Msr4ml: reconstructing artifact traceability in machine learning repositories. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 536–540. IEEE, 2021.

[19] D. Sato, A. Wider, and C. Windheuser. Continuous delivery for machine learning. *Martin Fowler*, 9, 2019.

[20] M. Schlegel and K.-U. Sattler. Management of machine learning lifecycle artifacts: A survey. *ACM SIGMOD Record*, 51(4):18–35, 2023.

[21] M. Tuape, V. T. Hasheela-Mufeti, A. Kayanda, J. Porras, and J. Kasurinen. Software engineering in small software companies: consolidating and integrating empirical literature into a process tool adoption framework. *IEEE Access*, 9:130366–130388, 2021.

[22] A. K. Wagner, S. B. Soumerai, F. Zhang, and D. Ross-Degnan. Segmented regression analysis of interrupted time series studies in medication use research. *Journal of clinical pharmacy and therapeutics*, 27(4):299–309, 2002.

[23] T. M. Ward, P. Mascagni, Y. Ban, G. Rosman, N. Padoy, O. Meireles, and D. A. Hashimoto. Computer vision in surgery. *Surgery*, 169(5):1253–1256, 2021.

[24] S. Wazir, G. S. Kashyap, and P. Saxena. Mlops: A review. *arXiv preprint arXiv:2308.10908*, 2023.

[25] M. Wessel, A. Serebrenik, I. Wiese, I. Steinmacher, and M. A. Gerosa. Quality gatekeepers: investigating the effects of code review bots on pull request activities. *Empirical Software Engineering*, 27(5):108, 2022.

[26] M. Wessel, J. Vargovich, M. A. Gerosa, and C. Treude. Github actions: the impact on the pull request process. *Empirical Software Engineering*, 28(6):1–35, 2023.

[27] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

[28] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu. The impact of continuous integration on other software development practices: a large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 60–71. IEEE, 2017.