# An Exploratory Study on Machine Learning Model Management

JASMINE LATENDRESSE, Concordia University, Canada
SAMUEL ABEDU, Concordia University, Canada
AHMAD ABDELLATIF, University of Calgary, Canada
EMAD SHIHAB, Concordia University, Canada

Effective model management is crucial for ensuring performance and reliability in Machine Learning (ML) systems, given the dynamic nature of data and operational environments. However, standard practices are lacking, often resulting in ad hoc approaches. To address this, our research provides a clear definition of ML model management activities, processes, and techniques. Analyzing 227 ML repositories, we propose a taxonomy of 16 model management activities and identify 12 unique challenges. We find that 57.9% of the identified activities belong to the maintenance category, with activities like refactoring (20.5%) and documentation (18.3%) dominating. Our findings also reveal significant challenges in documentation maintenance (15.3%) and bug management (14.9%), emphasizing the need for robust versioning tools and practices in the ML pipeline. Additionally, we conducted a survey that underscores a shift towards automation, particularly in data, model, and documentation versioning, as key to managing ML models effectively. Our contributions include a detailed taxonomy of model management activities, a mapping of challenges to these activities, practitioner-informed solutions for challenge mitigation, and a publicly available dataset of model management activities and challenges. This work aims to equip ML developers with knowledge and best practices essential for the robust management of ML models.

CCS Concepts: • **Software and its engineering** → *Maintaining software.*

Additional Key Words and Phrases: Software engineering, machine learning, model management.

**ACM Reference Format:**
Jasmine Latendresse, Samuel Abedu, Ahmad Abdellatif, and Emad Shihab. 2024. An Exploratory Study on Machine Learning Model Management. 1, 1 (August 2024), 32 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Machine Learning (ML) systems have seen tremendous growth over the years with increasing applications in healthcare, transportation, and finance, emerging as a cornerstone of technological advancement [1]. This evolution is intricately linked to advancements in deep neural network technologies, increased data availability, and enhanced access to computational resources [2], collectively forming the foundation for the evolution of ML systems. These pivotal elements enhance the capabilities and adaptability of ML systems, empowering them to effectively address complex problems across various domains [2, 3].

Authors' addresses: Jasmine Latendresse, Concordia University, Montreal, Canada, jasmine.latendresse@mail.concordia.ca; Samuel Abedu, Concordia University, Montreal, Canada, samuel.abedu@concordia.ca; Ahmad Abdellatif, University of Calgary, Calgary, Canada, ahmad.abdellatif@ucalgary.ca; Emad Shihab, Concordia University, Montreal, Canada, emad.shihab@concordia.ca.


ACM XXXX-XXXX/2024/8-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The dynamic nature of data and operational environments adds to the unique complexity of ML systems, necessitating models to be adaptable and robust against changing conditions and requirements [4]. Thus, practitioners need to effectively manage their ML models throughout their lifecycle as it directly impacts the performance and reliability of ML models [5]. In the literature, model management is defined as the process of tracking a model in all phases of its lifecycle [4]. This includes the initial development, where the model architecture and parameters are set; the training phase, which involves data handling and parameter tuning; deployment, where the model is integrated into its application environment; and monitoring and updating post-deployment. Specifically, this process involves tracking changes in model architecture, data sets, training parameters, performance metrics, and operational integration to effectively manage and optimize the model's reliability and performance.

However, prior work shows that organizations and practitioners often resort to ad hoc methods to implement the processes of model management, which can lead to inefficiencies and decreased reliability [6]. This not only creates logistical challenges but also poses risks in terms of model performance and applicability in real world scenarios. For example, inadequate model monitoring and updating procedures may result in inaccurate predictions or suboptimal outcomes, highlighting the importance of robust model management [6–8]. To the best of our knowledge, previous studies have not specifically discussed the tasks associated with model management. Identifying these tasks is essential to ensure that team members across different projects have a shared understanding of their responsibilities and workflows, ultimately improving model performance and reducing maintenance costs.

In their extensive review of existing literature, Nahar et al. emphasize the need for a structured approach to model management [8]. They argue that a clear definition of the activities, processes, and techniques involved in model management is crucial. This view is supported by Vartak et al., who define model management as the process of tracking a model in all phases of its lifecycle [5, 9]. The authors propose a tool to automate the processes of model management, focusing mainly on the implementation of the tool, keeping the decision-making and practical processes of model management black-box.

To address this gap and guide practitioners in effectively managing their models throughout the lifecycle, we build a taxonomy for model management activities. To achieve this, we mine commit messages and issues for 227 ML repositories from GitHub. Then we analyze and categorize various activities involved in model management and link them to specific challenges. To further enrich our findings, we conducted a survey with 21 practitioners from industry and academia, from which we observed a significant trend towards the automation of ML pipelines. Our work aims to address the following research questions:

**RQ1 What are the activities of machine learning model management?** We find six unique categories of model management activities, namely maintenance, development, environment, experimentation, data engineering, and deployment. Moreover, our findings indicate that a significant 57.9% of these activities fall under the maintenance category. This category encompasses tasks such as refactoring, documentation, bug fixing, and testing. Such a predominance of maintenance activities aligns with the dynamic nature of ML systems, which constantly need to adapt to changing data and requirements [4].

**RQ2 What are the challenges of machine learning model management?** Our study reveals 12 unique challenges of model management encompassing areas such as documentation maintenance, bug management, compatibility and dependency issues, and implementation difficulties. Notably, 15.3% of these challenges are associated with documentation maintenance. This is closely trailed by bug management at 14.9% and compatibility issues at 12.9%. These

findings suggest that practitioners often struggle with keeping documentation up-to-date and consistent with the frequent changes models are through.

**RQ3 How can machine learning model management challenges be mitigated?** Our survey results show a clear inclination among participants (47.6%) towards adopting tools (e.g., DVC) that enable versioning of models, data, and documentation. Additionally, 66.7% of respondents emphasize the importance of replicability by utilizing virtual environments and container technologies.

**Contributions.** In addition, to provide practitioners with a practical reference to efficiently allocate their efforts and resources, we analyze the complexity of model management activities using the time it takes to resolve issues. We believe that our study will assist practitioners in developing and managing models reliably and make the following contributions:

(1) We provide a taxonomy of 16 ML model management activities. To the best of our knowledge, this is the first study that provides a taxonomy of model management and its associated activities.
(2) We identify 12 unique challenges of model management that we map to model management activities.
(3) We conduct a survey with practitioners and provide practical insights into mitigating model management challenges.
(4) We provide a set of actionable recommendations based on our findings and experience in conducting this study for practitioners to guide them in managing their ML models.
(5) Based on our findings, we provide a definition of model management processes.
(6) We make our labeled data set of model management activities and challenges publicly available.[1]

**Paper Organization.** The rest of the paper is organized as follows. Section 2 describes our methodology. Section 3 presents the findings of our three research questions. Section 4 discusses the implications of our findings. Section 5 discusses related work. Section 6 outlines the threats to the validity of our study. Section 7 concludes this paper.

## 2 STUDY DESIGN

The main goal of our study is to identify the model management activities. To achieve this, we resort to analyzing ML projects on GitHub and examining ML commits in those projects. In this section, we describe the dataset used in our study and present the methodology to identify ML commits on these projects.

### 2.1 Dataset

To investigate how ML projects manage their models, we collect data from ML projects available on GitHub, which hosts over 350 million repositories. Also, GitHub allows to collect a rich dataset from open-source repositories with high traceability between commits, pull requests, and issues [10], which enables us to answer our research questions. We describe below the process we follow to curate the dataset for this study.

**Selecting candidate projects** For our analysis, we resort to the dataset proposed by Gonzalez et al. [1] and revised by Rzig et al. [11]. The dataset was initially created by using the GitHub API to curate a list of repository topic labels relevant to ML and then collecting projects with these labels. Rzig et al. then conducted a manual analysis of the dataset to remove trivial or non-ML

[1]https://zenodo.org/records/10602341

projects. The final set of projects contains 4,031 ML projects that are composed of ML frameworks and libraries such as Tensorflow,[2] as well as ML applications such as Faceswap.[3]

**Filtering candidate projects** To select projects for our study, we use the GitHub API to mine the project metadata of each project as of September 2023 and filter our dataset based on criteria already used in prior work [10, 12–14]:

- The repository has been in development for at least one year.
- The number of commits in the repository is at least 10.
- The number of days since the last push operation is at most 100.
- The repository has at least 100 stars.
- The repository has more than 3 contributors.

The choice of the above criteria is driven by several key considerations related to the objectives of our study, which centers on understanding established model management practices. First, repositories that have been active for at least a year are more likely to have reached a certain level of maturity and stability in their development processes and mature projects are more likely to have encountered and addressed various challenges, providing a richer source of data for our study. Second, repositories with at least 10 commits indicate a baseline level of development activity and engagement from contributors, suggesting that the project has moved beyond initial setup phases [14]. Third, limiting the selection to repositories with recent activity (last push not more than 100 days from the day of collecting the data) ensures that the selected projects are active and helps in studying the most current and relevant model management practices and challenges. Fourth, the number of stars on a repository serves as a proxy for its popularity and community approval. Thus, selecting repositories with at least 100 stars increases the likelihood that they are recognized by the community and that they incorporate established practices in their development [15]. Finally, including projects with more than three contributors ensures that the repository benefits from collaborative development, which may indicate that it has a well-defined workflow.

This filtering process yields a dataset of 365 ML projects. To further refine the dataset and ensure the relevance of the selected repositories in our study, we conducted a manual inspection of each repository. Specifically, we focused on the repositories' documentation, folder structure, and contained files, looking for evidence of training and testing code. During this process, we aimed to differentiate between general software projects that use "ML" or "AI" as keywords for broader purposes and those genuinely involved in ML workflows. For example, the project "akoumjian/-datefinder"[4] uses the keyword "NLP" but has no trace of any ML workflow in the repository.

For the purpose of our study, an "ML project" or "ML repository" is defined as one that directly involves the development or application of ML models, including but not limited to tasks like data preprocessing, model training, model evaluation, and the deployment of ML models. Projects solely providing libraries or frameworks that facilitate these activities without directly engaging in model lifecycle activities were classified separately and excluded if they did not demonstrate active use of these libraries in ML model processes within the repository itself.

Table 1 presents summary statistics for the key metrics of our final dataset comprised of 227 projects, including the number of commits, number of stars, open issues, time since the last push (as of September 2023), number of contributors, forks, watchers, repository size, and age of the repositories (measured in months). The statistics reveal that the projects of our dataset are active, with a median of 1,201 commits and a median of 6 days since the last push. Moreover, the median number of open issues stands at about 171, while the maximum is at 2,330. This indicates that some

[2]https://github.com/tensorflow/tensorflow
[3]https://github.com/deepfakes/faceswap
[4]https://github.com/akoumjian/datefinder

projects might struggle with issue resolution or have a highly active community. Also, the projects of our dataset are, on average, 69 months. Thus, our dataset consists of projects that are active, with high engagement, mature, with sustained development efforts. Moreover, we manually inspected the repositories of the selected projects and extracted various approaches and technologies that encompass both supervised and unsupervised learning methods. For instance, we found supervised learning techniques such as decision trees, neural networks, transfer learning, and deep learning, predominantly used for tasks requiring labeled data (e.g., knowledge graph classification[5] and drug discovery[6]). We also found unsupervised methods that include Bayesian networks and generative AI, which are more suited for tasks that involve pattern recognition[7] and data clustering[8] without pre-labeled examples.

Table 1. Descriptive statistics of the projects in our dataset.

| Statistic | Mean | Median | Std. Deviation | Minimum | Maximum |
|---|---|---|---|---|---|
| Nb Commits | 2898.04 | 1201.0 | 4944.04 | 51.0 | 32384.0 |
| Stars | 5204.94 | 2269.0 | 8500.80 | 130.0 | 64628.0 |
| Open Issues | 171.59 | 71.0 | 296.09 | 0.0 | 2330.0 |
| Time Since Last Push (days) | 19.52 | 6.0 | 25.92 | 1.0 | 99.0 |
| Contributors | 67.67 | 31.0 | 90.12 | 3.0 | 454.0 |
| Forks | 1416.86 | 431.0 | 4105.90 | 102.0 | 52849.0 |
| Watchers | 5204.94 | 2269.0 | 8500.80 | 130.0 | 64628.0 |
| Size (KB) | 120718.44 | 46244.0 | 207640.88 | 90.0 | 1345947.0 |
| Repo Age (months) | 69.76 | 62.53 | 24.31 | 32.73 | 161.37 |

### 2.2 Identifying ML Model Commits and Issues

In this section, we detail the filtering process used to identify commits and issues that involve changes related to ML models.

We start by mining the commits of the selected 227 repositories using Pydriller,[9] a Python framework to analyze Git repositories. For each commit, we collect the commit hash, commit message, number of insertions and deletions, and the list of files modified by the commit. This results in a total of 660,509 commits modifying 3,146,539 files. After a manual inspection of the dataset of the contents of sampled commits, we found commits with empty commit messages. We removed such commits from our dataset, yielding a final dataset of 659,773 commits, modifying 3,143,164 files.

To identify the commits performing changes on models, we use a set of heuristics. In particular, we randomly examined 100 repositories to identify the ML files in these repositories. We identified 14 unique keywords such as "models" and "train". Next, we search the file path and file name that contains these keywords or variations of different ML models such as "random forest", "random_forest", "randomforest". Prior works in ML have also utilized keyword-based searching [16, 17]. With this, we obtain a total of 95,765 commits modifying 386,936 model-related files, which serves as our primary source for empirical analysis in RQ1. Table 2 shows the complete list of keywords used in this step.

[5]https://github.com/pykeen/pykeen
[6]https://github.com/deepchem/deepchem
[7]https://github.com/jeeliz/jeelizFaceFilter
[8]https://github.com/jmschrei/pomegranate
[9]https://github.com/ishepard/pydriller

Table 2. Keywords used for filtering commit messages.

| Keyword | Variants |
|---|---|
| model | models |
| train | training |
| learn | learning, learner |
| regressor | regressors, regression |
| predict | predictor, predictors |
| linearregression | linear_regression, linear-regression |
| logisticregression | logistic_regression, logisticregression |
| randomforest | random_forest, randomforest |
| neuralnetwork | neural_network, neuralnetwork |
| supportvector | support_vector, supportvector, svm |
| naivebayes | naive_bayes, naive-bayes |
| knn | |
| kmeans | |

Similarly, we collect and filter the closed issues from the 227 ML projects of our dataset along with their meta-data, including titles, bodies, and pull request links. This step generates an extensive set of 376,758 issues, which is used for our analysis to answer RQ2.

### 2.3 Data Preparation for Categorization

The goal of RQ1 and RQ2 is to identify a set of ML model management activities (RQ1) and challenges (RQ2). For this, we resort to the manual categorization of the collected commits and issues from the 227 ML projects of our dataset. To obtain representative samples of commits and issues to categorize, we adopt a stratified random sampling technique. Specifically, we use a 95% confidence level with a 5% error margin, resulting in a sample size of 385 artifacts (either commits or issues) for each dataset. To guarantee the generalizability of our findings, we ensure that each project in our dataset is represented at least once. Next, we conduct a preliminary manual review of each sample to ensure the relevance of each data artifact to the topic of ML, addressing any instance that might have bypassed the initial filtering step. Should any item be deemed unrelated to ML (e.g., a web application[10] or a game engine[11]), we randomly replace it with another artifact to maintain the total sample count at 385. The descriptive statistics of the sample of commits and the sample of issues can be found in Table 3 and Table 4, respectively. Table 3 shows the number of deleted lines, the number of added lines, and the number of files affected by the commits. The median values for deleted and added lines are 15 and 33.5, respectively, which suggests that while most commits involve smaller changes, there are a few very large commits. Table 4 the number of comments, the number of assignees, the labels, and the age (in days) of the sampled issues. The mean and standard deviation for the number of comments per issue are of 3.1 and 4.2, respectively, which indicates that while some issues are highly discussed, the majority receive few comments. The number of assignees per issue has a median of 1, meaning that issues are typically assigned to one person, and the median age is of 2 days, indicating that most issues are resolved quickly. We detail the categorization process of the commits and issues in RQ1 and RQ2, respectively.

[10]https://github.com/guess-js/guess
[11]https://github.com/magefree/mage

Table 3. Descriptive Statistics of the Commit Sample

| | # Deleted Lines | # Added Lines | # Files |
|---|---|---|---|
| **Mean** | 1,808.6 | 2,510.5 | 28.7 |
| **Std. Deviation** | 26,902.8 | 27,026.7 | 267.2 |
| **Min** | 0 | 0 | 1 |
| **Median** | 15 | 33.5 | 4 |
| **Max** | 516,978 | 403,925 | 5,029 |

Table 4. Descriptive Statistics of the Issue Sample

| | # Comments | # Assignees | # Labels | Age (days) |
|---|---|---|---|---|
| **Mean** | 3.1 | 1 | 1.3 | 57 |
| **Std. Deviation** | 4.2 | 0.1 | 0.6 | 175.1 |
| **Min** | 0 | 1 | 1 | 0 |
| **Median** | 2 | 1 | 1 | 2 |
| **Max** | 34 | 2 | 5 | 1,462 |

### 2.4 Survey Design

In RQ3, we aim to identify solutions addressing the challenges highlighted in RQ2. For this, we conduct an online survey, grounding our methodology in the guidelines proposed by Kitchenham et al. [18]. We construct our survey into three distinct sections. The first section captures demographic insights, asking participants about their professional background and experience working with ML models. Subsequent sections in the survey (sections 2 and 3) present the challenges outlined in RQ2, prompting participants to pinpoint their strategies to mitigate each challenge. To avoid overwhelming participants, we provided potential solutions for each challenge. To do so, we systematically analyzed the sample of 385 pull requests' discussions and code changes associated with the studied issues. This helped us identify practical solutions that developers applied to resolve the raised issues. Furthermore, for issues that had no apparent fix, we extended our search to academic and industry-focused literature. We found a number of notable studies and articles, such as the study by Nahar et al., which reports a meta-summary of the most commonly reported machine learning challenges in surveys with industry practitioners [8], the study by Guan et al., which presents a comprehensive study of bugs in ML model optimization [19], and the article by Swimm Team, which discusses the concept of "Documentation as Code" in the context of software development [20]. Using these sources, we curated a list of potential solutions for each challenge, ensuring that our survey participants had access to a set of options to choose from and comment on. Moreover, to allow participants for more flexible answers, we facilitated multiple options for responses, incorporating an "Other" option for unlisted solutions, enabling participants to provide responses (solutions) that might not have been covered by the predefined choices. We also provide a "Does not apply" option in cases the participants did not encounter a particular challenge in their practice. We then pilot the survey with four practitioners and used their insights and feedback to refine the survey for better clarity. Next, we leverage our network connections in industry and academia to distribute it. For a more extensive distribution of our survey, we employ a snowball sampling technique, asking initial respondents to cascade it to other professionals in their network, particularly those with experience in the field of ML [21].

## 3 RESULTS

In this section, we present the results of our three research questions. For each research question, we present its motivation, the approach to answer the question, and the results.

### RQ1: What are the activities of ML model management?

**Motivation.** In the realm of machine learning, concrete model management processes are still poorly defined and often vary significantly across different teams and projects [22, 23]. This lack of standardization can lead to higher maintenance costs and reduced model performance, especially in diverse collaborative environments [6]. A fundamental step towards mitigating this issue is to clearly identify and define the activities that are part of the model management process. Thus, in this RQ, we establish a taxonomy of model management activities based on the analysis of open source ML repositories, aiming to provide a structured framework that practitioners can adopt. This will not only help in standardizing the tasks involved in model management, but also ensure that all team members of a project have a shared understanding of what needs to be done, how, and when.

**Approach.** To answer this research question, we categorize the sample of commits described in Section 2.3 into model management activities. r2c8, r3c7: To achieve this, we build an initial ML model management scheme using prior work [5, 6, 9, 24]. This list served as a base for our preliminary scheme, which we then expanded and refined through an analysis of the commit data. To refine our scheme, we employ an iterative process. The first two authors independently label 100 commits from the selected sample by analyzing the commits' code changes, messages, as well as any linked pull requests. Second, the annotators meet to compare their labeling and refine the scheme by adding, editing, or removing labels as they gain new insights from the data. Next, with the refined scheme, the authors then independently label the remaining 285 artifacts from the sample and measure their inter-rater agreement using the Cohen's Kappa coefficient, a well-known statistic used to measure the inter-rater agreement level for categorical scales. The resulting Kappa value is a scale that ranges between -1.0 and +1.0, where a negative value means poorer than chance agreement, zero indicates agreement by chance, and a positive value indicates better than chance agreement. The Kappa value after labeling the commits is 0.61, which indicates a substantial level of inter-rater agreement [25]. It is worth noting that the annotators agreed that in some cases, multiple labels should be applied to one commit message even if it is considered "atomic". Thus, one instance of the sample may be classified with more than one label. However, since the Kappa assumes that the labels are mutually exclusive (i.e., one label per instance of the sample), we transformed the multi-label data into multi-instances before calculating the Kappa coefficient.[12] Any discrepancies or disagreements in the labeling are resolved through discussion and consensus among the authors. Finally, activities that are related to a common phase of model management are grouped into broader categories. For example, the commit message "refactoring parameter handling",[13] classified as a refactoring activity, and the commit message "Fix JUnitsRUnitsPyUnits after recent change to shuffling of training data per chunk"[14] classified as a testing activity, are both grouped under the "maintenance" category because they aim to enhance the model's reliability and performance without introducing new features or functionalities.

**Results.** Table 5 shows the results of our manual classification of GitHub commit messages, yielding a total of 16 model management activities grouped under 6 categories. From this, **activities under `Maintenance` category account for over half (57.9%) of the total activities**, revealing

[12] https://docs.kolena.com/metrics/tp-fp-fn-tn/

[13] https://github.com/elki-project/elki/commit/3c3ded2c7667f03e4c502477733550322a99d00c

[14] https://github.com/h2oai/h2o-3/commit/7c13f3bd7c4f4cb8fd2bf56afc989f6f423cd55c

Table 5. Definition and distribution of model management activities.

| | Activity | Definition | % of Occ. | |
|---|---|---|---|---|
| **Maintenance** | **Refactoring** | Enhancing the clarity, structure, and maintainability of the model's codebase without changing its functional behavior. | 20.5% | 57.9% |
| | **Documentation** | Detailing the model's capabilities and performance, offering guidelines for use, and providing illustrative examples and tutorials. | 18.3% | |
| | **Bug Fix** | Addressing and rectifying unintended behaviors or issues in the model's code and implementing safeguards and responses within the code to manage and address potential runtime anomalies and issues. | 11.6% | |
| | **Testing** | Evaluating the correctness, behavior, and compatibility of model components, ensuring they work as intended. | 7.5% | |
| **Development** | **Model Behavior Enhancement** | Augmenting an existing model by introducing new features, optimizing existing functionalities, or boosting its overall performance. | 10.6% | 18.9% |
| | **Model Design** | Defining and developing the structure, layers, and organization of the machine learning model. | 3.9% | |
| | **Functional Configurations** | Adjusting settings that dictate the model's input and output behaviors, such as input types and processing capabilities. | 2.5% | |
| | **Model Versioning** | Cataloging and preserving different iterations of a model, encapsulating specific combinations of its code, data, and configurations for reproducibility and tracking. | 1.9% | |
| **Environment** | **Dependency Management** | Keeping track of and updating external machine learning libraries and tools that the model relies on. | 3.7% | 10.3% |
| | **Plumbing** | Handling behind-the-scenes system-related tasks essential for the model's operation but not directly related to its ML logic, such as crafting command-line interfaces. | 3.5% | |
| | **Compatibility Management** | Ensuring the model can seamlessly operate and interact within its designated environment, including specific operating systems and external utilities. | 3.1% | |

| | | | | |
|---|---|---|---|---|
| **Experiment** | **Training and Validation** | The process of training the model on datasets and subsequently evaluating its predictive accuracy and effectiveness using standard metrics, including de- termining the best set of hyperparameters for the model to achieve optimal results and performance. | 5.4% | 7.7% |
| | **Experiment Logging** | Capturing and recording important model-related metadata, error messages, and runtime events for diagnostic and tracking purposes. | 2.3% | |
| **Data Engineering** | **Data Engineering** | Preparing, transforming, and organizing data to be fit for model training and validation. | 4.1% | 4.1% |
| **Deployment** | **CI/CD** | Automating processes associated with building, integrating, and deploying the model to operational settings. | 0.6% | 1% |
| | **Model Porting** | Translating or adapting a trained model so it can function across diverse computing platforms or environments. | 0.4% | |

the importance of maintaining the health of machine learning models post-development. The `Maintenance` category is primarily concerned with the health and clarity of a machine learning model after it is deployed into production. It covers a range of activities from documentation to bug fixes to ensure the continued operability, readability, and reliability of the model's codebase. This suggests that once a model is developed, ensuring its continued utility and performance becomes the central focus.

Specifically, 20.5% of the activities in our dataset, where developers enhance the clarity of the model's codebase through code refactoring. For example, the commit message "Move NeuralNetClassifier and -Regressor to own modules"[15] encompasses various refactoring tasks. The primary goal is to enhance the codebase's maintainability, modularity, and comprehensibility. This is achieved by relocating ML components to dedicated modules, restructuring tests to align with the changes, and removing unused functions and imports. Thus, it ensures that the code remains agile and can adapt to new requirements without incurring a heavy technical debt [26]. *Documentation* emerges as the second most frequent activity under `Maintenance` (and overall) with 18.3%, which indicates that practitioners place an emphasis on describing model functionalities and efficacy. We find that documentation-related tasks do not only pertain to keeping up-to-date information on the model, but also provide example use-cases through Jupyter Notebooks.[16] Those examples serve as a quick start guide for users to use and integrate the ML model in their application. These examples serve as quick-start guides, enabling users to efficiently integrate and utilize the ML model in their applications. Thus, practitioners maintain the examples in parallel with deployed models to ensure its reproducibility, as it is seen in the commit "Updated GAN model notebook".[17]

[15] https://github.com/skorch-dev/skorch/commit/271f422c7bda4af8bbc44a3388a7d2e3bd285f90
[16] https://jupyter.org/
[17] https://github.com/deepchem/deepchem/commit/55d1ff20ad5583f88aeedca012529d8e15f820aa

The Maintenance category also includes *bug fixing*, accounting for 11.6% of all activities. Our analysis of the labeled commit messages and code changes reveals that bugs range from minor implementation issues (e.g., calling incorrect tokenizer[18]) to more serious bugs that could compromise model results and performance (e.g., selecting the wrong channel for loss multiplier[19]). This leads to another critical activity of model management, *testing*, which accounts for 7.5% of all activities. *Testing* is the process of ensuring that models not only operate as intended but can also perform across varied and unforeseen scenarios (e.g., a model designed for facial recognition might struggle under unusual lighting conditions, or skin types that weren't prevalent in the training data, leading to biases). Robust testing protocols ensure the model's reliability, performance, and its ability to make precise predictions. It is also a proactive step to minimize future bugs and issues [27, 28].

**Development activities account for a significant portion of model management (18.9%)**, but are less predominant than Maintenance. These activities are foundational as they pertain to the creation, design, and improvement of machine learning models. Within this category, *model behavior enhancement*, which consists of refining existing models with additional functionalities to adapt to evolving requirements, is the most prevalent activity under Development with 10.6%. For example, the commit "relocating functors, adding inline to device functors, adding entropy for classification"[20] describes reorganizing and optimizing device-specific functions, while also enhancing the classification process by considering the entropy of predictions. These changes contribute to enhancing the model's performance and ability to adjust or accommodate to changing data or evolving requirements. Interestingly, we note that the occurrences of *Model Design* (3.9%) and *Functional Configurations* (2.5%) activities are relatively low compared to *Model Behavior Enhancement*, as shown in Table 5. These activities include the definition and configuration of the model's structure and layers, exerting a direct impact on the model's performance. One possible reason for this observation is that such activities occur locally (i.e., not committed to the project's repository). Finally, the last activity of the Development category, *model versioning* which accounts for 1.9% of all occurrences, is an activity focused on recording and saving various iterations of a model. While *model versioning* is not as predominant as other activities, it remains crucial for ensuring that specific combinations of its code, data, and configurations are traceable and can be replicated [9].

**About 10.3% of the activities are linked to the system and external configurations within which a machine learning model is deployed and functions.** The Environment activities include updating external ML libraries and performing system related tasks (e.g., updating the code to accommodate API changes[21]). Those activities ensure that the ML model can operate seamlessly and consistently across various environments (e.g., Windows, MacOS). Understanding the environment settings is critical as it helps practitioners develop reproducible and scalable models, avoid dependency conflicts, and mitigate the risk of unexpected issues that might arise during production [9].

**The Experiment category accounts for 7.7% of the activities in our dataset.** The Experiment category is centered around the processes of tuning, testing, and refining machine learning models to ensure their accuracy and effectiveness. Before publishing the ML model, practitioners often experiment with the model to evaluate the model's performance to understand how well the model is likely to perform in real-world scenarios. Moreover, Experimentation allows practitioners to identify and address errors, bugs, or anomalies present in the model.

[18] https://github.com/dmlc/gluon-nlp/commit/46e77acf2561d9680c36c9f93e1db19a8f8de575
[19] https://github.com/deepfakes/faceswap/commit/3852b2b2d1e19cd2f0d72d111e9d6733bbf5ae26
[20] http://github.com/rapidsai/cuml/commit/229bd038b691a47e2eca9392065705d1729a942c
[21] https://github.com/mne-tools/mne-python/pull/9274/commits/e4c75db77890e70a7c5760645deedc41e40e61e1

Data is the backbone of any ML model due to its fundamental role in shaping the model's effectiveness, performance, and reliability. In our dataset, the *Data engineering* activities account for 4.1% of activities. These activities pertain to data preprocessing carried out prior to feeding it into the model during both the training and serving stages. (e.g., adding new features to the training data.[22] We believe that the low occurrences of data engineering activities in our dataset might be due to the fact the data are stored in data versioning control (e.g., DVC and MLFlow).

The `Deployment` category accounts for only 1% of total occurrences, making it one of the least frequent category of activities. Within this category, *CI/CD* (i.e., continuous integration and deployment) processes contribute 0.6%, focusing on automation associated with model deployment, while *model porting*, which deals with making the model functional across various platforms, constitutes 0.4%. The lower frequency of these activities does not imply lesser importance. Instead, the limited occurrences can be attributed to the nature of deployment activities, which are often conducted at the system level (e.g., DevOps) involving cloud, server, and extensive tool configurations, which are typically not committed to the project's repository. These activities are specialized and might not occur as frequently as other activities, but they remain critical for operationalizing models in real-world environments [9].

Based on the findings presented in Table 5, we define machine learning model management as a continuous process that encompasses activities aimed at ensuring the efficiency, reliability, and effectiveness of ML models throughout their lifecycle. This process includes the inception and development of the model's architecture, and its continuous refinement, maintenance, and adaptation to meet changing requirements and operational environments. Specifically, model management covers:

- **Maintenance activities**: Refactoring for codebase clarity and maintainability; comprehensive documentation detailing the model's use and functionalities; bug fixes to address and correct anomalies; and testing to ensure correct behavior and compatibility of model components.
- **Development activities**: Model behavior enhancement through the introduction of new functionalities or performance optimization; functional configurations to dictate input-output behaviors; model versioning for tracking changes; and dependency management to maintain external library and tool integrations.
- **Environment setup**: Plumbing to handle essential system-related tasks that support the model's operation; and compatibility management to ensure seamless model operation across various systems.
- **Experimentation**: Training and validation of models to evaluate accuracy and performance; and experiment logging to record essential metadata for future evaluation.
- **Data engineering**: The preparation and transformation of data to suit model training and validation.
- **Deployment**: Continuous integration and continuous deployment (CICD) processes for building and integrating models; and model porting to adapt models for use across different platforms or environments."

Hence, model management in the context of machine learning is the interdisciplinary coordination of the aforementioned activities to ensure that models remain functional, effective, and adaptable throughout their lifecycle - from initial implementation to deployment and after.

[22]https://github.com/openvenues/libpostal/commit/6a20ce5e854a4a0c347b8da887eff511ac849ca5

Machine learning model management spans a wide range of activities, with maintenance being the most dominant category of activities, grouping 57.9% of all activities. This suggests that there is a significant focus on clarity, maintainability, and documentation in real-world projects post-development. Particularly, *refactoring* (20.5%) and *documentation* (18.3%), stand out as a vital insight for teams managing and scaling machine learning models in collaborative settings. The `Development` phase (18.9%) and `Environment` compatibility (10.3%) of models also emerged as critical aspects of model management.

**RQ2: What are the challenges of model management?**

**Motivation.** Building upon the taxonomy of model management activities outlined in RQ1, this RQ focuses on the complexities that have arisen due to the rapid advancements and widespread application of AI and ML technologies. These complexities hinder practitioners' ability to effectively manage their models [6, 8]. By identifying and analyzing the challenges currently faced in model management, this RQ aims to highlight potential pitfalls and issues that practitioners encounter. Understanding these challenges in the current context of machine learning is pivotal not only for improving the efficiency and reliability of ML models but also for helping practitioners in mitigating risks associated with ad hoc, custom practices of model management [8, 22, 23]. Thus, this RQ contributes towards establishing more systematic and effective model management processes by providing insights into particularly challenging areas, in which best practices can be applied.

**Approach.** To address this RQ, one of the authors starts by selecting GitHub issues relevant to model management by manually reviewing each issue's title, body, comments, and labels. Issues primarily associated with non-model management tasks were excluded from the sample and replaced. Next, to identify the challenges of ML model management, we employ the open card coding process to categorize the sample of issues described in Section 2.3 based on the issue title, body, pull request, and comments. In particular, the authors developed a preliminary coding scheme based on a preliminary analysis of the sampled issues, with a particular focus on identifying the main challenges. Then, the first author initiates the process with "open coding", reading through the issues to identify and note emerging themes, patterns, or challenges from which a coding scheme is developed. This scheme consists of categories or "codes" that represent the identified themes. For example, in the context of ML model management, codes might represent specific types of challenges, such as data quality and computational resource constraints. To ensure that the codes are agreed on by all authors, the coding scheme then undergoes an iterative refinement process involving four key steps. After the initial coding is developed by the first author, all authors meet to discuss the preliminary codes. This step aims to gather feedback on the relevance, clarity, and comprehensiveness of the initial codes. Based on the feedback, codes are modified, merged, or split to better capture the challenges observed in the issues. With each iteration of modifying the coding scheme, a subset of issues is re-coded to test the applicability of the revised codes. The iterative process continues until no changes are made to the codes and all authors agree on the final set of codes.

**Results.** Table 6 shows the results of our manual classification of GitHub issues, yielding a total of 13 model management challenges. From this, we see that **the most prominent challenge is the necessity for `Documentation Maintenance`, constituting about 15.3% of the total challenges**. This challenge highlights the difficult in updating of documentation to mirror model modifications, including new feature integrations, bug fixes, and performance metric adjustments. Moreover, maintaining up-to-date tutorials and example models form a critical part of this challenge, since the final model is often the result of many different experimentations including multiple

rounds of data processing, hyperparameter tuning, and other activities as described in RQ1. Thus, despite the critical importance of documentation for the model's reproducibility and maintainability, developers find it difficult to reflect the model's continuous changes in a way that is accessible for stakeholders of different levels [6]. This finding aligns with Nahar et al.'s work on challenges in building systems with machine learning components, in which they report that "practitioners find documentation more important than ever in ML, but find it more challenging than traditional software documentation"[8].

**Following closely is `Bug Management`, comprising 14.9% of occurrences**. This challenge highlights the difficulty in identifying, tracking, and rectifying bugs that might hinder the model or its components from functioning as intended (e.g., inconsistent exceptions occuring during training with specific metrics.[23] `Bug Management` is particularly challenging in the context of machine learning since models, instead of failing, will simply give incorrect results, and the lack of testing oracles make such "silent failures" difficult to identify. `Bug Management` also involves an efficient feedback loop between users and developers given the frequent back and forth on collaborative platforms when reporting and resolving bugs. This interaction also involves participants with widely varying levels of experience, yet often lacks a systematic approach to identifying and resolving issues [6, 8]. These results further corroborate with our findings from RQ1, which identified maintenance-related activities (*documentation* and *bug fix*) as some of the most prominent activities of model management. The need for continuous documentation updates- covering features, performance metrics, and example models- underscores the rapidly evolving nature of machine learning models. Similarly, the considerable attention to bug tracking and fixing highlights the complexity and dynamic nature of machine learning systems.

**`Compatibility` issues, accounting for 12.9% of all occurrences, reflect the challenges in ensuring that the machine learning model or its components are aligned with various operational environments such as different operating systems or programming language versions.** These challenges stem from the experimental nature of model development; having to transition from an experimental environment, like exploratory code, often in a notebook, to a wider, pipeline-driven system that integrates both ML and non-ML components as well as many tools and frameworks [8, 29, 30]. Additionally, machine learning models, especially deep learning models, often have specific hardware requirements such as particular GPU architectures and ensuring the compatibility across different environments necessitates to be proactive in keeping up with updates and changes in the entire pipeline, which often requires a wide array of knowledge that is not only focused on machine learning components. This is further reiterated by Vartak et al. who proposed ModelDb, a tool to address the challenge of heterogeneity in models as well as their hardware and software requirements [9].

**`Dependency` issues, accounting for 11.2% of all occurrences, involve handling the external dependencies that influence the flexibility and performance of a model**. Dependencies in a project can range from libraries and frameworks to system tools and external services that the project relies upon. What makes it particularly challenging in the context of model management is that machine learning systems have both ML and non-ML components, which creates multiple levels of dependencies to manage. The first and most distinct form of dependency in models is data. Machine learning systems are fundamentally dependent on data pipelines for training and inference and the dynamic nature of data, such as changes in data distribution and structure, makes it unique to model management [31]. Library and frameworks dependencies are also crucial in model management since machine learning projects often rely on a myriad of libraries and frameworks that are constantly updated, meaning that models need frequent adjustments to

[23]https://github.com/RubixML/ML/issues/64

stay compatible with the latest versions, making the dependency management more challenging compared to traditional systems. Furthermore, machine learning models incorporate a wide range of algorithms that encompass various optimization techniques, statistical methods, or neural network architectures, each with their own dependencies that evolve, adding a level of complexity not commonly found in traditional systems [8].

**Our analysis further reveals that 10.1% of the encountered challenge are attributed to `Implementation` issues, involving potential shortcomings in the existing model structure and code**. As models continuously evolve with changing requirements, this challenge generally manifests as a deficiency in critical functional features or the accumulation of technical debt, thereby requiring regular refactoring as reported in RQ1. This corroborates with prior works reporting that ML-centric systems go through frequent revisions more than traditional systems and that practitioners struggle planning for change in their model implementation [2, 8, 21, 32–34]. **These issues can in turn lead to `Model Performance` challenges, which represent 7% of the challenges**. These challenges are associated with the prediction quality of models and includes tasks like benchmarking, managing overfitting and underfitting, adjusting hyperparameters, and constant monitoring of the model's performance. Practitioners report having difficulty with ad-hoc monitoring practices and that there is a severe lack of tooling to automate the monitoring process, making it difficult to integrate it to the pipeline early, which results in intensive manual work post-deployment [33, 35–37].

**The challenges associated with `Data Management`, which constitute 5.7% of all challenges, are critical in model management as data serves a foundation for model training, validation, and testing**. `Data Management` challenges span across the entire model development lifecycle and encompass issues with data acquisition, ensuring the data's integrity and reliability over time, data preprocessing, and even extends to data storage and retrieval. Such tasks mostly take place locally, which explains why `Data Management` challenge are not as frequent in our dataset despite being so critical to model management. Our analysis reveals that practitioners struggle with ensuring data quality and often resort to manual processing of the data (e.g., relying on an annotation tool for ML practitioners that does not label sentence pairs,[24] which further corroborates with previous works reporting that data quality is not well supported by tools and that practitioners need to invest significant effort and time in data preprocessing and cleaning [2, 8, 38, 39]. Also, the dynamic nature of model management creates the need for data versioning since it allows to maintain a historical record of data alterations, allowing to revert to previous versions or facilitating debugging by answering questions such as "Which data change made the model performance deteriorate?". However, the problem is that there is little tool support to automate such a task, which affects the traceability and reliability of the data [34].

**`Code Performance`, accounting for 4.8% of the reported challenges**, involves refining the source code to enhance speed and resource efficiency, focusing on areas such as algorithm improvement, computational efficiency, memory management, and leveraging optimal hardware utilization. As previously mentioned, the experimental nature of model development forces practitioners to transition from exploratory code, often in a notebook, to a wider, pipeline-driven system, which often lacks standardized review processes [36, 40, 41]. **`Resource Constraints` (3.7%) on the other hand, revolve around navigating the limitations in computational resources such as processing capacity, memory, storage, and bandwidth**. These challenges often manifest in training durations, storage capacity issues, latency problems, scalability, and considerations regarding cost-effectiveness (e.g., reducing `YOLOv5` post-processing time in a neural

[24] https://github.com/doccano/doccano/issues/24

network inference framework[25]). Previous works show that practitioners often have difficulties in building scaleable pipelines and adhering to serving requirements like latency and throughput [8, 21, 33, 34, 42]. Our analysis also reveals that `Code Performance` and `Resource Constraints` are not isolated challenges but rather closely linked aspects in model management. Optimized code performance can significantly alleviate resource constraints by ensuring more efficient utilization of computational power, memory, and storage.

**Our analysis also identifies `Infrastructure`, `Testing`, and `Modularity` as critical yet somewhat lesser represented challenges, comprising 3.3%, 2.4%, and 2.2% of the recorded challenges, respectively**. `Infrastructure` challenges highlight the issues of integrating and deploying models within functional systems, necessitating scalable infrastructure and systematic model monitoring. However, practitioners report that the infrastructure and tooling that supports model development is often lacking as it is not automated enough, is too difficult to integrate in the existing environment, and is too limited to specific tasks or data types [8, 37, 43]. Concurrently, `Testing`, which involves establishing robust tests to ascertain model behavior and accuracy, is a critical aspect of model management as it ensures the reliability and efficacy of the models, yet practitioners find it difficult to achieve due to the lack of quality assurance criteria and testing oracles. Nahar et al. explain that no model is expected to be always correct, but that it is difficult to define what kind of mistakes are acceptable for a model [8]. Moreover, there is a growing need for tools and automated test input generation to reduce the cost of curating test data. Meanwhile, `Modularity` issues hint at an emerging challenge in creating more manageable and efficient machine learning systems. However, the many implicit data and tooling dependencies created when designing systems that incorporate both ML and non-ML components make it difficult to modularize, reinforcing "change anything changes everything" [26]. While the aforementioned challenges are less prominent in our dataset, they are of significant importance as they shed light on the emerging issues that are faced by practitioners in ML projects.

Finally, 6.5% of the challenges fall under the `Other` category, which encompasses challenges that don't fit directly into the aforementioned classification and account for less than 1% or less of the total occurrences. Such challenges range from deciphering unclear error messages to ensuring model validation accuracy and streamlining deployment processes. This suggests that there is a wide range of model management challenges, some of which being less prominent but may be equally as crucial in specific contexts that occur outside collaborative platforms.

> Our findings reveal `Documentation Maintenance` (15.3%) and `Bug Management` (14.9%) as the most prominent challenges in model management, highlighting the necessity for constant up-to-date documentation and effective bug tracking throughout the model lifecycle. `Compatibility` (12.9%) and `Dependency` (11.2%) challenges underscore integration and system complexities, emphasizing the need for tools and automation in the ML pipeline to support dynamic model maintenance and optimization.

### RQ3: How can model management challenges be mitigated?

**Motivation.** In RQ2, we identified 12 key challenges of model management. While these challenges provide valuable insights, addressing them effectively is crucial for providing practitioners with actionable solutions. Therefore, RQ3 focuses on identifying concrete solutions and best practices currently employed by ML practitioners to navigate these challenges. Our aim is to compile and disseminate these actionable solutions, and provide the community with a set of tools and

[25] https://github.com/Tencent/ncnn/pull/3648

Table 6. Mapping model management challenges to model management categories.

| Challenge | Definition | % of Occ. |
|---|---|---|
| **Documentation Maintenance** | The continuous task of updating documentation to reflect model changes, including new features, fixes, and performance metrics, while also keeping tutorials and example models current. | 15.3% |
| **Bug Management** | The process of detecting, tracking, and rectifying bugs to ensure the model or ML component functions as expected. | 14.9% |
| **Compatibility** | Ensuring the model or ML component aligns with its operational environment, such as specific operating systems or programming language versions. | 12.9% |
| **Dependency** | Handling external dependencies that might impact the model's flexibility and performance, including updates, integrations, and addressing vulnerabilities. | 11.2% |
| **Implementation** | Dealing with the model or ML component's present structure, particularly if it lacks essential features or induces technical debt. | 10.1% |
| **Model Performance** | Addressing issues tied to prediction quality, encompassing benchmarking, overfitting, underfitting, hyperparameter adjustments, and ongoing monitoring. | 7% |
| **Data Management** | Overseeing the collection, organization, and processing of data for model training, validation, and testing. | 5.7% |
| **Code Performance** | Optimizing the source code for speed and resource efficiency, focusing on algorithmic enhancement, computational efficiency, memory management, and optimal hardware use. | 4.8% |
| **Resource Constraints** | Navigating limitations in computational resources, like processing capacity, memory, storage, and bandwidth. Challenges span training durations, storage capacity, latency, scalability, and cost-effectiveness. | 3.7% |
| **Infrastructure** | Integrating and deploying models into functional systems. This includes setting up infrastructure, scaling, system integration, and ongoing model monitoring. | 3.3% |
| **Testing** | Instituting rigorous tests to validate model behavior and accuracy, addressing issues like inconsistent tests, the lack of test benchmarks and oracles, and managing extensive input data. | 2.4% |
| **Modularity** | Segmenting the model and ML components into distinct modules, which may require reorganizing dependencies or incorporating new modules for comprehensive solutions. | 2.2% |
| **Other** | Encompasses issues such as deciphering ambiguous error messages, ensuring model validation accuracy, organizing project structures for efficiency, guaranteeing model reproducibility, streamlining deployment processes, maintaining consistent code styling, and navigating potential license conflicts. | 6.5% |

methodologies that have been proven effective. This will not only serve as a practical blueprint for mitigating the identified challenges but also guide practitioners in adopting standardized practices in model management. By providing these solutions, we enable practitioners to learn from others, and ensure that they can implement effective strategies in their own processes.

**Approach.** To identify solutions addressing the challenges discussed in RQ2, we conduct an online survey as described in Section 2.4. Recognizing that the length of a survey (i.e., number of questions) can deter participation [18], we decide to cluster challenges into groups. Specifically, we group challenges tied to `Compatibility` and `Infrastructure`, `Implementation` and `Modularity`, as well as `Resource Constraints` and `Code Performance` into clusters. This led to a more concise design encompassing 9 questions, representative of 12 challenges. Upon distributing our survey, we received a total of 21 responses.

Table 7 provides a breakdown of the demographics of the 21 participants with varying background and levels of experience with ML systems. A majority of the participants are students (52.4%), industry practitioners make up 38.1% of the sample, and researchers represent 9.5% of the participants. Moreover, the majority of the participants have at least two years of experience with ML systems, ensuring diversity across different levels of experience and backgrounds. It is worth noting that we deliberately reached out to students to conduct our survey for a few specific reasons. First, we knew that most of these students have experience in developing machine learning systems for at least a year (72.7%). Second, expecting busy industry developers to dedicate significant time to our study would be challenging and would limit the depth of their insights and thus, our analysis. Also, prior research suggests have shown that experienced students can effectively represent professional developers, particularly when focusing on new technologies, as is the case in our study [44, 45]. Thus, the real value for this study does not lie in the characteristics of the participants, but rather in the insights they provide to build a set of actionable solutions to concrete model management challenges.

**Results.** Table 8 presents the challenges, solutions derived from the literature, and the distribution of selected solutions by participants. As we mentioned in the approach of this research question, participants had the flexibility to opt for multiple answers for each question, and propose a solution of their own. We marked the participant-proposed solutions with an asterisk ("*").

From the table, we observe that there is no single superior solution for each challenge. In other words, practitioners employ various solutions to overcome the specified challenges. **Overall, the results show that there is a widespread preference for using *hyperparameter optimization techniques* (66.7%) for `Model Performance` challenges**. Hyperparameter optimization is essential for achieving peak model performance [46], emphasizing the significance of fine-tuning these parameters to optimize and enhance model outcomes in tackling these challenges.

**Our results also show a repeated reference to tools and practices that ensure replicability**. For example, 66.7% and 52.4% of participants selected *virtual environments* and *using containers*; respectively. These results highlight the growing importance of maintaining isolated ML environments, a sentiment shared by practitioners [47]. One possible reason for the prevalence of virtual environments among practitioners is to ensure the reproducibility of the ML model regardless of the underlying system. Moreover, 47.6% of respondents also agreed that using data versioning tools (e.g., DVC and MLFlow) to keep track of data changes was a viable solution to the data management challenges. In addition to contributing towards reproducible ML, these tools underscore a trend toward the automation of the ML pipeline, reflecting the industry shift towards DevOps principles, where continuous integration, delivery, and deployment are critical [48].

Conversely, the survey results reveal that not all pre-listed solutions applied to the experiences of the participants. Notably, the least chosen solution, with no participants advocating for it, is in the `Bug Management` challenges, which is the construction of a function mapping between code in

different programming languages. The need for a function mapping across different programming languages is dependent on the development stack and practices of a given project or organization as it relies on static analysis of low-level code [19]. For participants of our survey, their work might not span multiple programming languages, or the potential complexity of such mappings may not justify the benefits within their operational contexts. Another potential reason for this observation is that organizations resort to develop ML applications using Python as it provides many OSS libraries such as Scikit-Learn, TensorFlow, and PyTorch [49]. Another uncommon solution (4.8% of respondents) is the use of efficient memory allocation for `Resource Constraints` challenges. Efficient memory allocation involves planning the use of memory based on when data is needed during training. The unpopularity of this solution may be due to the fact that ML tasks often deal with diverse and dynamic data. The memory required can vary significantly based on the size and complexity of the dataset or model being used. Thus, predicting and allocating the precise amount of memory needed can be challenging.

**Similarly, We find that certain challenges might not be experienced by all participants, as they selected the *"Does not apply"* options various challenges.** Interestingly, 23.8% of participants do not face challenges related to `Documentation Maintenance` and `Bug Management`. However, in our manual examination of the issues in RQ2, we find that these two challenges are the most frequently encountered. The main reasons for this observation are 1) the RQ2 results originate from OSS projects, where project maintainers prioritize comprehensive documentation to encourage tool usage and contributions [50]. In fact, some OSS projects (e.g., Qiskit Machine Learning[26] and Scikit-Learn[27]) require contributors to update documentation before accepting their contributions. 2) 52% of the participants, comprising both graduate and undergraduate students, might not prioritize documentation in their projects as much as in public projects. Despite this, documentation remains a critical aspect of model management, especially in collaborative settings, as was shown by our results and previous work [51–53].

In light of this, we provided participants with the latitude to propose their own solutions. **This lead to the emergence of seven solutions that were not previously accounted for in our suggested solutions**. For instance, *updating documentation with each release* is a participant-proposed solution for the `Documentation Maintenance` challenges and highlights the oft-neglected commitment of keeping documentation aligned with the model's evolution. Similarly, solutions such as *applying noise-handling algorithms* for `Data Management` and *using strict semantic versioning* to avert dependency conflicts reflect insights into challenges that may not be universally experienced but are critical in certain contexts. Furthermore, the suggestion to *batch-process data in small chunks* to overcome resource constraints may be effective in environments with limited computational capacity. Another participant-proposed solution of *validating service data with schema-based validation* for `Bug Management` challenges allows for catching bugs related to data quality early in the ML pipeline, which corroborates with the work of Breck et al. [31].

> The results of our survey highlight a clear inclination among participants towards integrating tools for versioning data, models, and documentation. Participant-proposed solutions highlight the industry's rapid evolution, underscoring the need for replicability through the use of virtual environments and containers.

[26] https://github.com/qiskit-community/qiskit-machine-learning/blob/main/CONTRIBUTING.md
[27] https://scikit-learn.org/dev/developers/contributing.html

Table 7. Background of participants in the survey.

| Dimension | Experience | % |
|---|---|---|
| Background | Student | 52.4% |
| | Industry Practitioner | 38.1% |
| | Researcher | 9.5% |
| ML Systems | 1-2 years | 38.1% |
| | 3-5 years | 33.3% |
| | <1 year | 23.8% |
| | >5 years | 4.8% |
| **Total participants** | | **21** |

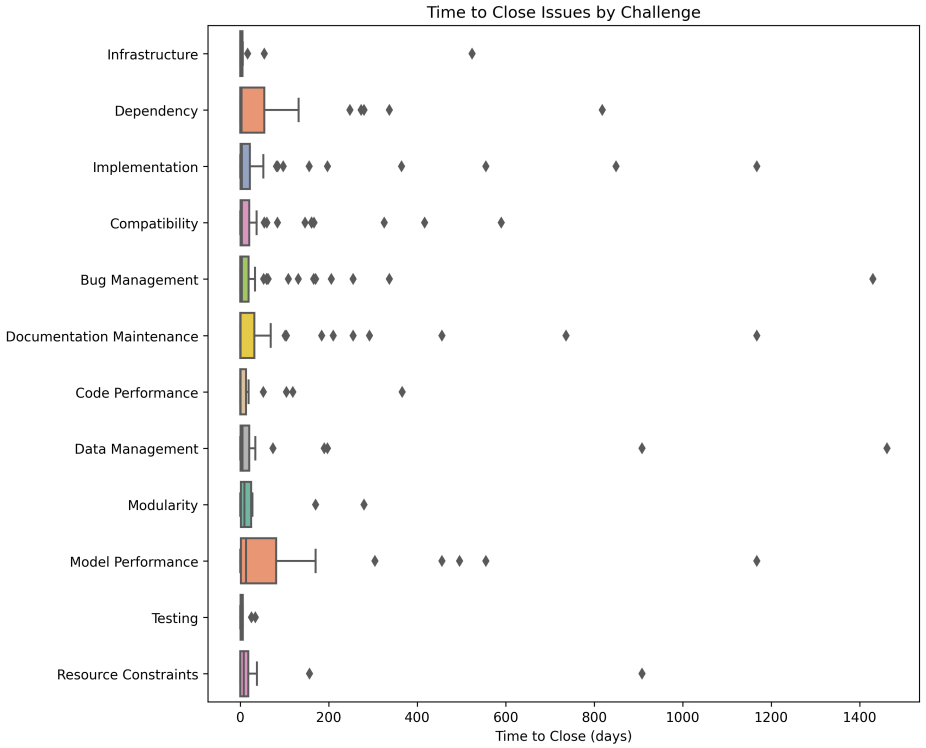


Fig. 1. Time Distribution to close GitHub Issues by Challenge.

## 4 DISCUSSION

In this section, we delve into the evaluation results to gain deeper insights into the complexity of activities. Additionally, we provide a set of actionable recommendations for practitioners and researchers to guide them in their ML model management.

Table 8. Survey results for identifying solutions to the model management challenges. Participants were free to select more than one solution per challenge. Annotated statements (*) represent new solutions suggested by participants.

| Challenge | Solution | % of Occ. |
|---|---|---|
| **Documentation Maintenance** | Reviewing documentation as part of code review [20]. | 61.9% |
| | Publishing documentation as part of CI/CD to ensure example notebooks are not broken [20]. | 28.6% |
| | Versioning documentation as you version models [20]. | 19% |
| | Updating documentation with each release* [20]. | 4.8% |
| | Does not apply. | 23.8% |
| **Data Management** | Using data versioning tools to keep track of data changes [7]. | 47.6% |
| | Including data anomaly detection in the data preprocessing pipeline [31, 54]. | 42.9% |
| | Data augmentation (i.e., artificially increasing the size of training data) [55]. | 23.8% |
| | Apply noise-handling algorithms*. | 4.8% |
| | Does not apply. | 23.8% |
| **Bug Management** | Adopting metamorphic testing (i.e., verify test results consistency with multiple inputs) [56]. | 61.9% |
| | Using standard test suites to mimic common bugs (i.e., faultload benchmarks) [57]. | 28.6% |
| | Validate service data with schema-based validation*. | 4.8% |
| | Creating a function mapping between code in different programming languages to aid static code analysis tools in detecting bugs [19]. | 0% |
| | Does not apply. | 23.8% |
| **Model Performance** | Using hyperparameter optimization techniques to find optimal hyperparameter values that yield the best performance [46]. | 66.7% |
| | Continuously monitor model performance in production environment to detect and fix model drift [58]. | 52.4% |
| | Implement transfer learning [59]. | 19% |
| | Exercise caution when partitioning data into training, testing, and validation sets*. | 4.8% |
| | Does not apply. | 14.3% |
| **Dependency Management** | Using virtual environments to prevent dependency conflict with other services [47]. | 66.7% |
| | Selecting dependencies that are self-contained and avoid relying on a large number of dependencies [26]. | 33.3% |
| | Using dependency management tools (e.g., Dependabot) [60]. | 19% |
| | Using strict semantic versioning to avoid breaking changes*. | 4.8% |
| | Does not apply. | 14.3% |

| | | |
|---|---|---|
| **Compatibility & Infrastructure** | Using containers (e.g., Docker) to deploy the model and keep environment consistent [60]. | 52.4% |
| | Implement auto-scaling (i.e., adjusting resources based on demand) to manage varying levels of system solicitation at different times [61]. | 38.1% |
| | Implementing CI/CD to run on one or multiple operating systems and integrating continuous training (CT) [48, 62]. | 23.8% |
| | Monitor the resource usage of the model and use CI/CD for the one OS (i.e., the deployment server)*. | 4.8% |
| | Does not apply. | 23.8% |
| **Implementation & Modularity** | Following design patterns for ML applications (e.g., ensuring reproducibility by setting seeds or using checkpoints) [26]. | 47.6% |
| | Implement system functionalities into independent modular components [63]. | 42.9% |
| | Using versioning tools (e.g., DVC and MLFlow) to keep track of data and configurations [7]. | 38.1% |
| | Re-training existing models on new features to learn corrections, as opposed to creating cascading models [26]. | 28.6% |
| | Does not apply. | 19% |
| **Resource Constraints & Code Performance** | Optimize training and inference code to reduce unnecessary computations [64]. | 47.6% |
| | Perform distributed training (i.e., parallelizing model training across multiple machines) [65]. | 38.1% |
| | Use transfer learning and/or incremental learning techniques to reduce computational load [59]. | 19% |
| | Batch processing of small chunks*. | 4.8% |
| | Efficient memory allocation [64]. | 4.8% |
| | Does not apply. | 28.6% |
| **Testing** | Use different types of tests to evaluate different functionalities of the system [66]. | 52.4% |
| | Use statistical tests (e.g., Kolmogorov-Smirnov test) to detect data drift [67]. | 23.8% |
| | Integrate test coverage metrics [66]. | 19% |
| | Does not apply. | 19% |

### 4.1 The Complexity of Model Management Activities

In RQ2, we identified the challenges associated with model management activities. The complexity varies across these challenges due to differences in technical requirements, varying degrees of expertise needed to address them, and the diverse environments in which models are deployed and operated. Understanding the complexity of these challenges can aid practitioners in prioritizing their work by considering the areas where the most difficult challenges arise when managing their

models. Therefore, in this section, our goal is to gauge the complexity of each challenge. For this, we measure the time taken to resolve the GitHub issues used to identify the challenges in RQ2. This metric has been used in previous studies to assess the complexity of resolving issues [68–70].

Figure 1 presents the closure time distributions for each challenge. From the figure, it's evident that challenges specific to machine learning (ML), such as `Model Performance` and `Data Management`, exhibit among the longest closure times. For example, issues related to `Model Performance` take a median time of 13 days to close. ML performance-related issues often have deep-seated roots within the model's architecture, algorithms, or the data used in its training. Identifying the exact cause of a performance decline necessitates delving into them, which can be time-consuming. Moreover, rectifying ML performance issues often demands a more comprehensive approach. This involves retraining the model using refined datasets, adjusting its hyperparameters, or reconsidering the choice of algorithm, and ensure that the solution enhances a model's performance requires rigorous testing and validation, which is also a time-intensive process.

Unsurprisingly, `Data Management` stands out as one of the most complex challenges in terms of closure time. This is because data serves as the backbone of any ML system, and the model's quality is fundamentally rooted in the quality of the data used for its training. Consequently, practitioners invest significant time in preprocessing and cleaning the data to guarantee its high quality [39]. Our findings suggest that the research community should intensify efforts in developing approaches and tools that aid practitioners in managing their models, especially in enhancing their performance.

Another challenge that is of significant complexity is `Resource Constraints`, which is underscored by the presence of outliers indicating prolonged resolution times. ML workflows are resource-intensive and require substantial computational power and memory, often exacerbated by the size of datasets and complexity of models. In particular, the prolonged resolution times of `Resource Constraints` may reflect challenges that are often faced by open-source project contributors who rely on consumer-grade hardware. In such context, contributors to open-source ML projects may seek to optimize performance within the constraints of limited computational resources. Notably, an outlier that persisted for 908 days, dealt with an inefficiency in data storage causing significant storage overhead.[28] The issue in question was ultimately closed with a resolution of "unlikely to be fixed". Our findings shed light on the need for resource-efficient ML implementations and tools that are accessible to a broader audience.

### 4.2 Implications for Practitioners

Based on our analysis and findings of each research question, we list the implications for practitioners to help them better manage their ML models.

**Develop Centralized Platforms that Facilitate Collaboration.** The results of RQ2 and RQ3 show that there is a significant gap in the ecosystem for centralized, collaborative platforms that facilitate comprehensive model management and experiment tracking. Our findings are aligned with prior work that emphasized the necessity of tracking a model across all phases of its lifecycle, including detailed logging of metrics for training and validation [5, 6]. Yet, our analysis uncovered very minimal evidence of experiment tracking within the studied projects. This might stem from the inherent characteristics of the ML workflow, but it highlights a critical challenge. GitHub, despite its widespread use for hosting open-source ML projects, does not inherently facilitate the tracking and sharing of experimental workflows essential for ML development. Experiments in ML are frequently run locally and, as we observed during the labeling process in RQ1, result in large, infrequent commits that obscure the developmental trajectory of models when viewed on traditional collaborative platforms [39]. To address these challenges, we recommend that developers

[28] https://github.com/BVLC/caffe/issues/1321

and organizations: (1) engage with collaborative platforms in a more granular manner, such as by making smaller, more frequent commits, allowing for a clearer and more continuous history of model evolution, and (2) ensure that detailed documentation of each model iteration is integrated and regularly synchronized with model modifications and versions.

**Emphasize and Prioritize Robust Documentation Practices.** Effective documentation is one of the cornerstones of a collaborative ML environment as it records each iteration of the model, every decision made, and the rationale behind changes. This not only facilitates transparency but also aids in attracting new contributors, providing them with a clear history of the model's development [51–53]. Despite the importance of documentation, our findings of RQ2 show that maintaining it is among the most challenging activities of model management. Thus, we recommend practitioners incorporate documentation as an integral part of the development process, not as an afterthought. This can be achieved by: (1) establishing documentation protocols and standards that detail what needs to be documented at each stage of the model lifecycle (e.g., documentation needs to be present with every pull request), (2) utilizing version control systems to track changes in documentation alongside code changes to ensure that they are always in sync, and (3) establishing documentation as a core component of the model management process, akin to coding and testing.

**Implement Resource Allocation Strategies for Complex ML Tasks.** Our analysis in RQ2 reveals unpredictability in issue resolution times (ranging from less than a day to 1,462 days), particularly for complex ML tasks from which emerged significant outliers, impacting overall project timelines and resource planning [69]. Proper resource allocation is important for managing complex ML tasks to improve project timelines and resource planning. In other words, by allocating additional time and resources to complex tasks like data management, practitioners can more effectively anticipate and address potential delays in key areas of their ML applications, allowing for a more efficient allocation of resources. This not only pertains to human resources but also to the management of computational resources, which are often crucial in collaborative and open-source environments as it can directly affect the reproducibility of experimental results [51].

### 4.3 Implications for Researchers.

Based on our analysis and findings of each research question, we list the implications for researchers to guide future investigations into challenging areas of ML model management.

**The Need for Improved Documentation Strategies.** In RQ1, we find that the role of documentation-related activities emerges as particularly significant. In RQ2, we pinpoint documentation maintenance as one of the most challenging aspects of model management. The consistent emphasis on documentation across different studies, including our study, highlights the need for deeper investigation into the underlying reasons for this challenge from the research community [51–53]. For example, why maintaining accurate and up-to-date documentation is especially problematic in this field? What unique obstacles do ML practitioners face when updating documentation? How do the dynamic nature of ML models and the frequent updates to data sets complicate documentation efforts?

Answering these questions is crucial for developing more effective documentation strategies that are tailored to the complexities of ML projects. The recent explosion of Large Language Models (LLM) and their success in different software engineering tasks presents a unique opportunity for researchers to study the potential of these technologies in enhancing documentation practices. By studying the automation of the documentation process with LLMs, researchers can help ensure that documentation evolves in tandem with model and data changes, minimizing discrepancies and enhancing model management efficacy for practitioners.

**Develop Automated Compatibility Check.** One of the most recurring model management challenges identified in RQ2 is ensuring that the model or ML component aligns with its operational environment, such as specific operating systems or programming language versions. This is particularly significant given how rapidly software and hardware environments are evolving. To address this, researchers should focus on automating the ML pipeline by creating automated frameworks that can simulate various deployment scenarios and enhancing versioning tools with a compatibility requirements tracker alongside model updates.

**Managing External Dependencies in ML Projects.** Unlike traditional software, ML systems often integrate a diverse array of data sources, libraries, and tools, each of which may evolve independently [8]. This creates unique challenges of dependency management, where changes in one element can affect the stability of the entire system. This complexity significantly affects the reliability and performance of ML models, especially due to their non-modular nature, as observed in RQ2. To help practitioners, future research should focus on developing methodologies that allows to isolate components within ML systems. Such methodologies should include the adoption of containerization technologies (e.g., Docker) to encapsulate dependencies, and the use of microservices architecture to modularize processes.

**Enhancing Model Reproducibility.** In RQ1 and RQ2, we highlight that discrepancies in model versioning, experiment tracking, and documentation are obstacles to replicating and validating results of ML models. In RQ3, the use of versioning tools was underscored by respondents as potential solutions for data management and modularity challenges. However, currently available tools do not scale to the complexity of ML systems [71, 72]. Thus, empirical studies should be conducted to evaluate and enhance the efficacy and capabilities of versioning tools in the context of ML systems.

## 5 RELATED WORKS

In this section, we discuss the existing literature related to model management approaches and challenges.

### 5.1 Model Management and Challenges

Previous studies have laid foundational work in the field of model management [4–6, 8, 73–75]. Vartak et al. [5] define model management as the process of tracking a model across all phases of its life cycle. They discuss some challenges of model management that include a lack of standardized practices and agreed upon methods amongst practitioners. To address these challenges, the authors propose ModelDB, a tool to automate model management tasks [9]. Andrei et al. [4] discussed challenges specific to deploying ML in production and stages of ML model deployment. The problem is that the definition of model management remains unclear and that the processes implemented by model management tools is black-box.

Schelter et al. [6] define model management as "the training, maintenance, deployment, monitoring, organization, and documentation of ML models" and identify challenges associated with these processes in four selected use cases: time series forecasting, missing value imputation, content moderation, and automating model metadata tracking [6]. Their discussion categorized the challenges into three groups: conceptual challenges, which pertain to the lack of clear definitions and standards in the field; data management challenges, which focuses on issues about abstractions used in the ML pipelines, and engineering challenges, which involve technical complexities of implementing and maintaining ML systems. Notably, the authors highlight the challenge of model validation, discussing the fact that machine learning models are dynamic and highly depend on data, methods, and dependencies. As such, a single change in one of components of the system leads to re-validating the model's performance, which corroborates with our findings in RQ2,

where we discuss the challenges of isolating changes given the non-modular nature of ML systems. Furthermore, the authors report on the importance of re-evaluating the model's performance with the same training, test and validation set, which we further emphasize in the results of RQ2 pertaining to the challenges of documenting changes in ML pipelines. However, while this work provides a foundational understanding of model management challenges, their analysis is primarily rooted in the context of four specific ML use cases while ours focuses on a larger, more diverse set of projects. This study provides valuable insights, but may not fully capture the broader range of challenges encountered in newer ML environments that have emerged with recent technological advancements in the field. Our work builds upon and extends their framework by developing a comprehensive taxonomy of ML model management activities, based on the analysis of practical, real world ML projects. Additionally, we offer practitioner-informed solutions for each identified challenge, something that Schelter et al. [6] touched upon but did not expand fully.

Chen et al. further refine the challenges of deploying ML-based software by presenting a taxonomy of ML development challenges [73]. In comparison, our study focuses on challenges specific to model management, which have not been clearly defined in the current literature.

Building upon insights from Isbell et al. [76], who discuss the adaptation of established software engineering practices for ML systems, our study integrates these practices within our taxonomy, addressing the critical need for standardized procedures in ML model management. Isbell et al. [76] emphasize the importance of integrating software engineering basics such as testing, documentation, and version control into the ML workflow, which aligns with our findings on the challenges in documentation maintenance and the need for robust versioning tools.

Moreover, extending the discussion on the black-box nature of Ml systems by Cao et al. [77], our work contributes to understanding the explainability within model management tasks. Cao et al. highlight the challenges posed by the opaque decision-making processes in ML-based systems, which underscores the need for explainable approaches that are crucial for validating and auditing ML models. This is especially true in collaborative environments, and directly informs the design of our taxonomy.

Furthermore, the paper by Wang et al. [78]emphasizes the impact of machine and deep learning on software engineering practices by cataloging multiple applications ranging from code analysis to project management. This integration of ML in SE highlights the urgent need for robust model management practices that not only handle the lifecycle of ML models but also adapt to the changing requirements of software engineering processes, which our work aims to address by proposing model management strategies.

More recently, Nahar et al. conducted an extensive literature review, providing a meta-summary study of 50 papers involving 4,758 practitioners to report on the most commonly encountered challenges in the field of ML [8]. Notably, the authors report that ML adds substantial complexity with many, often implicit data and tooling dependencies, and entanglements due to a lack of modularity, which further corroborates with our findings of RQ2. Moreover, the authors observe that the development of ML systems is often ad hoc, lacking well-defined processes, which further motivates the need for a clear and structured taxonomy of model management processes.

### 5.2 Automation in ML Pipelines

Tools and approaches for model management can be found in the literature. Vartak et al. proposed ModelDB, the first open-source model management system [9]. Chen et al. present MLflow, a popular open-source platform for managing the development of ML components [79]. Repositories using MLflow tend to expose issues related to ML model management in each phase of the machine learning lifecycle. Weber et al. propose MMP, a model management tool specific for Industry 4.0 environments [80].

Idowu et al. present a feature-based survey of 18 state-of-practice and 12 state-of-research tools supporting ML asset management [24]. The authors discuss that developing ML systems requires the management of a greater variety of asset types than traditional systems, including models. Also, the authors argue that ML asset management is an essential discipline when building ML systems, which further motivates our goal to focus on model management.

The problem is that while tools and frameworks exist to automatically manage models, there is still a lack of understanding and agreement on well-defined model management practices. Thus, in this work, we offered a software engineering perspective on ML model management and proposed a taxonomy of model management activities. We also highlighted task specifications and challenges related to model management and provide practical solutions to the identified challenges by surveying industry practitioners and academic researchers.

## 6 THREATS TO VALIDITY

**Threats to internal validity** considers the experimenter's bias and errors. Our method of analysis largely depends on manual inspection and labeling. Thus, a potential threat to the validity of our study is the subjective nature of categorizing the model management activities. This subjectivity might affect the reproducibility of our classification schema. To mitigate this risk, two authors independently categorized the activities and calculated the Cohen's Kappa coefficient and achieved substantial agreement. Discrepancies in labeling were meticulously resolved through consensus among all authors.

Our method of identifying commits that modify models is based on a keyword list derived from related literature, expert knowledge, and a thorough manual review of the structure of the repositories from our dataset. This approach carries the risk of not encompassing all relevant project structures, potentially overlooking some source files containing model code. To address this, our keyword list includes various forms of ML terminologies, such as "randomforest", "random_forest", and "randomforest". Furthermore, we have implemented a manual validation process to ensure that the files identified indeed contain model source code by inspecting the source code of each file in the selected commits.

Furthermore, in our sampling of GitHub issues, we excluded those perceived as unrelated to ML models and components. However, there is no guarantee that all categorized issues pertain exclusively to ML components. To mitigate this, each issue's associated pull request and comments were manually reviewed and analyzed for relevance.

**Threats to External validity** considers the generalizability of the findings. Our study focuses on open-source projects hosted on GitHub, which may not fully represent the wider spectrum of projects, especially proprietary ones. To mitigate this, we employed established best practices in mining software repositories, selecting active, popular, and mature ML projects [12, 13]. Moreover, our dataset contains projects that have already been studied in the literature and encompass a diverse range of ML projects, ensuring the inclusion of well-known repositories like Facesweap and Tensorflow [11]. This diverse selection aims to broaden the applicability and relevance of our findings across different ML project contexts.

Moreover, we surveyed 21 participants to gain insights into the practical challenges and solutions in model management as discussed in RQ3. Thus, the participants might not capture the extensive range of experiences and opinions in the field of model management. Thus, we made concerted efforts to include a wide diversity of participants, from industry practitioners to academic researchers to ensure a broad spectrum of perspectives Furthermore, our survey results might be more reflective of the views and experiences of those who chose to participate, rather than the entire population of ML professionals. To mitigate this, we designed our survey with predefined options, while also allowing respondents the opportunity to share their personal insights or indicate

if the provided options were not relevant to their specific expertise. Also, while less than half have over five years of experience, this reflects the evolving nature of the field and the growing adoption of ML by practitioners with diverse backgrounds. We acknowledge that experience can be a factor influencing perceptions of model management challenges. Future research could specifically target senior-level developers to investigate potential differences in model management practices among various experience levels.

Another potential threat to the external validity of our findings is related to the saturation of the taxonomy of model management activities we have proposed. Particularly, whether the inclusion of additional repositories might lead to the emergence of new categories within our taxonomy. Nevertheless, the used ML projects in our study are used in similar prior work [11, 81, 82]. Furthermore, we encourage future research to re-evaluate and expand our taxonomy with new data sources or as new practices become prevalent.

## 7 CONCLUSION

In this work, we presented an extensive investigation into the management of ML models, identifying key activities and challenges in this domain. Our analysis, grounded in the manual inspection and labeling of the commits of 227 ML repositories, reveals the dynamic nature of model management and underscores the necessity for standardized practices in this field, especially in collaborative settings. We proposed a taxonomy of 16 model management activities, which offers clarity and structure to a field that has, until now, largely relied on ad hoc approaches. This taxonomy, curated from analyzing real-world projects, provides a framework that can be applied in both academic and industrial contexts for the effective management of ML models. Our findings also highlight the growing need for automating the ML pipeline, particularly emphasizing the importance of versioning for data, models, and documentation. In line with this, we observed a significant focus on replicability, with participants in our survey shedding light on the value of virtual environments and containers in achieving consistent and reliable ML model management and deployment. Based on our observations, we make several recommendations, notably that when using collaborative platforms such as GitHub, developers should keep their commits atomic and push changes more incrementally, allowing for more reproducible models. Recognizing the connection of ML model management with broader ML software management, future work should also explore the evolution and management of ML components within software systems to extend our taxonomy to cover these aspects. Furthermore, we plan (and encourage others) to develop approaches and tools to automate the model management pipeline and enable better monitoring of model changes within collaborative platforms. This is important because understanding the connection between model management and broader software management can lead to more comprehensive and effective strategies for maintaining ML systems. Our study is the first step toward achieving this, laying the groundwork for future research and development in this area.

## REFERENCES

[1] D. Gonzalez, T. Zimmermann, and N. Nagappan, "The state of the ml-universe: 10 years of artificial intelligence & machine learning software development on github," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 431–442.

[2] 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)*. IEEE, 2019, pp. 291–300.

[3] M. Dilhara, A. Ketkar, and D. Dig, "Understanding software-2.0: A study of machine learning library usage and evolution," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–42, 2021.

[4] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," *ACM Comput. Surv.*, 2022.

[5] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia, "Modeldb: a system for machine learning model management," in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, 2016, pp. 1–3.

[6] S. Schelter, F. Biessmann, T. Januschowski, D. Salinas, S. Seufert, and G. Szarvas, "On challenges in machine learning model management," 2015.

[7] S. Idowu, D. Strüber, and T. Berger, "Asset management in machine learning: State-of-research and state-of-practice," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–35, 2022.

[8] N. Nahar, H. Zhang, G. Lewis, S. Zhou, and C. Kästner, "A meta-summary of challenges in building products with ml components–collecting experiences from 4758+ practitioners," *arXiv preprint arXiv:2304.00078*, 2023.

[9] M. Vartak and S. Madden, "Modeldb: Opportunities and challenges in managing machine learning models." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 16–25, 2018.

[10] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining github," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, 2016.

[11] D. E. Rzig, F. Hassan, C. Bansal, and N. Nagappan, "Characterizing the usage of ci tools in ml projects," in *Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2022, pp. 69–79.

[12] G. Gousios and D. Spinellis, "Mining software engineering data from github," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 501–502.

[13] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating github for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, 2017.

[14] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 92–101. [Online]. Available: https://doi.org/10.1145/2597073.2597074

[15] H. Borges and M. T. Valente, "What's in a github star? understanding repository starring practices in a social coding platform," *Journal of Systems and Software*, vol. 146, pp. 112–129, 2018.

[16] S. Biswas, M. Wardat, and H. Rajan, "The art and practice of data science pipelines: A comprehensive study of data science pipelines in theory, in-the-small, and in-the-large," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2091–2103.

[17] S. Dutta, A. Shi, R. Choudhary, Z. Zhang, A. Jain, and S. Misailovic, "Detecting flaky tests in probabilistic and machine learning applications," in *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*, 2020, pp. 211–224.

[18] B. A. Kitchenham and S. L. Pfleeger, *Personal Opinion Surveys*. London: Springer London, 2008, pp. 63–92. [Online]. Available: https://doi.org/10.1007/978-1-84800-044-5_3

[19] H. Guan, Y. Xiao, J. Li, Y. Liu, and G. Bai, "A comprehensive study of real-world bugs in machine learning model optimization," in *Proceedings of the International Conference on Software Engineering*, 2023.

[20] S. Team, "Documentation as code: Why you need it &; how to get started," Aug 2023. [Online]. Available: https://swimm.io/learn/code-documentation/documentation-as-code-why-you-need-it-and-how-to-get-started

[21] A. Serban and J. Visser, "Adapting software architectures to machine learning challenges," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 152–163.

[22] C. Hill, R. Bellamy, T. Erickson, and M. Burnett, "Trials and tribulations of developers of intelligent systems: A field study," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2016, pp. 162–170.

[23] S. Hillemacher, N. Jäckel, C. Kugler, P. Orth, D. Schmalzing, and L. Wachtmeister, "Artifact-based analysis for the development of collaborative embedded systems," in *Model-Based Engineering of Collaborative Embedded Systems*. Springer, 2021, pp. 315–331.

[24] S. Idowu, D. Strüber, and T. Berger, "Asset management in machine learning: State-of-research and state-of-practice," *ACM Comput. Surv.*, 2022. [Online]. Available: https://doi.org/10.1145/3543847

[25] J. L. Fleiss and J. Cohen, "The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability," *Educational and Psychological Measurement*, vol. 33, no. 3, pp. 613–619, 1973.

[26] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," *Advances in neural information processing systems*, vol. 28, 2015.

[27] S. Herbold and T. Haar, "Smoke testing for machine learning: simple tests to discover severe bugs," *Empirical Software Engineering*, vol. 27, no. 2, p. 45, 2022.

[28] H. B. Braiek and F. Khomh, "On testing machine learning programs," *Journal of Systems and Software*, vol. 164, p. 110542, 2020.

[29] H. Liu, S. Eksmo, J. Risberg, and R. Hebig, "Emerging and changing tasks in the development process for machine learning systems," in *Proceedings of the international conference on software and system processes*, 2020, pp. 125–134.

[30] S. Zdanowska and A. S. Taylor, "A study of ux practitioners roles in designing real-world, enterprise ml systems," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–15.
[31] E. Breck, N. Polyzotis, S. Roy, S. Whang, and M. Zinkevich, "Data validation for machine learning." in *MLSys*, 2019.
[32] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *2018 44th euromicro conference on software engineering and advanced applications (SEAA)*. IEEE, 2018, pp. 50–59.
[33] M. Haakman, L. Cruz, H. Huijgens, and A. van Deursen, "Ai lifecycle models need to be revised. an exploratory study in fintech," *arXiv preprint arXiv:2010.02716*, 2020.
[34] W. Hummer, V. Muthusamy, T. Rausch, P. Dube, K. El Maghraoui, A. Murthi, and P. Oum, "Modelops: Cloud-based lifecycle management for reliable and trusted ai," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019, pp. 113–120.
[35] G. A. Lewis, I. Ozkaya, and X. Xu, "Software architecture challenges for ml systems," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 634–638.
[36] S. Shankar, R. Garcia, J. M. Hellerstein, and A. G. Parameswaran, "Operationalizing machine learning: An interview study," *arXiv preprint arXiv:2209.09125*, 2022.
[37] A. Bäuerle, Á. A. Cabrera, F. Hohman, M. Maher, D. Koski, X. Suau, T. Barik, and D. Moritz, "Symphony: Composing interactive interfaces for machine learning," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–14.
[38] V. Golendukhina, V. Lenarduzzi, and M. Felderer, "What is software quality for ai engineers? towards a thinning of the fog," in *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, 2022, pp. 1–9.
[39] M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "Data scientists in software teams: State of the art and challenges," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1024–1038, 2017.
[40] N. Nahar, S. Zhou, G. Lewis, and C. Kästner, "Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 413–425.
[41] Z. Wan, X. Xia, D. Lo, and G. C. Murphy, "How does machine learning change software development practices?" *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1857–1871, 2019.
[42] A. Serban, K. van der Blom, H. Hoos, and J. Visser, "Adoption and effects of software engineering best practices in machine learning," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–12.
[43] M. Saidur Rahman, F. Khomh, A. Hamidi, J. Cheng, G. Antoniol, and H. Washizaki, "Machine learning application development: Practitioners' insights," *arXiv e-prints*, pp. arXiv–2112, 2021.
[44] I. Salman, A. T. Misirli, and N. Juristo, "Are students representatives of professionals in software engineering experiments?" in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 666–676.
[45] M. Höst, B. Regnell, and C. Wohlin, "Using students as subjects—a comparative study of students and professionals in lead-time impact assessment," *Empirical Software Engineering*, vol. 5, pp. 201–214, 2000.
[46] M. Feurer and F. Hutter, "Hyperparameter optimization," *Automated machine learning: Methods, systems, challenges*, pp. 3–33, 2019.
[47] A. Grigorev, "Deploying machine learning models, part 3: managing dependencies," Jun 2021. [Online]. Available: https://freecontent.manning.com/deploying-machine-learning-models-part-3-managing-dependencies/
[48] I. Karamitsos, S. Albarhami, and C. Apostolopoulos, "Applying devops practices of continuous automation for machine learning," *Information*, vol. 11, no. 7, p. 363, 2020.
[49] OpenSourceForU, "Why python is popular for machine learning implementations," 2021, accessed: 2023-11-09. [Online]. Available: https://www.opensourceforu.com/2021/02/why-python-is-popular-for-machine-learning-implementations/
[50] R. Watson, M. Stamnes, J. Jeannot-Schroeder, and J. H. Spyridakis, "Api documentation and software community values: a survey of open-source api documentation," in *Proceedings of the 31st ACM international conference on Design of communication*, 2013, pp. 165–174.
[51] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCunn, K.-R. Muller, F. Pereira, C. E. Rasmussen *et al.*, "The need for open source software in machine learning," 2007.
[52] W. Bangerth and T. Heister, "What makes computational open source software libraries successful?" *Computational Science & Discovery*, vol. 6, no. 1, p. 015010, 2013.
[53] Z. Yang, C. Wang, J. Shi, T. Hoang, P. Kochhar, Q. Lu, Z. Xing, and D. Lo, "What do users ask in open-source ai repositories? an empirical study of github issues," *arXiv preprint arXiv:2303.09795*, 2023.
[54] S. García, J. Luengo, F. Herrera, S. García, J. Luengo, and F. Herrera, "Data preparation basic models," *Data Preprocessing in Data Mining*, pp. 39–57, 2015.
[55] C. Shorten, T. M. Khoshgoftaar, and B. Furht, "Text data augmentation for deep learning," *Journal of big Data*, vol. 8, pp. 1–34, 2021.

[56] M. Leotta, D. Olianas, and F. Ricca, "A large experimentation to analyze the effects of implementation bugs in machine learning algorithms," *Future Generation Computer Systems*, vol. 133, pp. 184–200, 2022.
[57] M. M. Morovati, A. Nikanjam, F. Khomh, and Z. M. Jiang, "Bugs in machine learning-based systems: a faultload benchmark," *Empirical Software Engineering*, vol. 28, no. 3, p. 62, 2023.
[58] Jun 2023. [Online]. Available: https://developer.nvidia.com/blog/a-guide-to-monitoring-machine-learning-models-in-production/
[59] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.
[60] B. John, "Ml model packaging [the ultimate guide]," May 2023. [Online]. Available: https://neptune.ai/blog/ml-model-packaging
[61] M. Imdoukh, I. Ahmad, and M. G. Alfailakawi, "Machine learning-based auto-scaling for containerized applications," *Neural Computing and Applications*, vol. 32, pp. 9745–9760, 2020.
[62] [Online]. Available: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning
[63] S. Mittal, Y. Bengio, and G. Lajoie, "Is a modular architecture enough?" *Advances in Neural Information Processing Systems*, vol. 35, pp. 28 747–28 760, 2022.
[64] J. Zhang, S. H. Yeung, Y. Shu, B. He, and W. Wang, "Efficient memory management for gpu-based deep learning systems," *arXiv preprint arXiv:1903.06631*, 2019.
[65] S. Mittal and S. Vaishay, "A survey of techniques for optimizing deep learning on gpus," *Journal of Systems Architecture*, vol. 99, p. 101635, 2019.
[66] D. Marijan, A. Gotlieb, and M. K. Ahuja, "Challenges of testing machine learning based systems," in *2019 IEEE international conference on artificial intelligence testing (AITest)*. IEEE, 2019, pp. 101–102.
[67] S. Ackerman, O. Raz, M. Zalmanovici, and A. Zlotnick, "Automatically detecting data drift in machine learning classifiers," *arXiv preprint arXiv:2111.05672*, 2021.
[68] S. Panichella, G. Canfora, and A. Di Sorbo, ""won't we fix this issue?" qualitative characterization and automated identification of wontfix issues on github," *Information and Software Technology*, vol. 139, p. 106665, 2021.
[69] R. Kikas, M. Dumas, and D. Pfahl, "Using dynamic and contextual features to predict issue lifetime in github projects," in *Proceedings of the 13th International Conference on Mining Software Repositories*, 2016, pp. 291–302.
[70] N. Bühlmann and M. Ghafari, "How do developers deal with security issue reports on github?" in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022, pp. 1580–1589.
[71] Y. Zhao, "Mlops and data versioning in machine learning project," 2020.
[72] A. T. Njomou, M. Fokaefs, D. F. Silatchom Kamga, and B. Adams, "On the challenges of migrating to machine learning life cycle management platforms," in *Proceedings of the 32nd Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '22. USA: IBM Corp., 2022, p. 42–51.
[73] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu, "A comprehensive study on challenges in deploying deep learning based software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 750–762.
[74] P. Rani, S. Panichella, M. Leuenberger, A. Di Sorbo, and O. Nierstrasz, "How to identify class comment types? a multi-language approach for class comment classification," *Journal of systems and software*, vol. 181, p. 111047, 2021.
[75] P. Rani, A. Blasi, N. Stulova, S. Panichella, A. Gorla, and O. Nierstrasz, "A decade of code comment quality assessment: A systematic literature review," *Journal of Systems and Software*, vol. 195, p. 111515, 2023.
[76] C. Isbell, M. L. Littman, and P. Norvig, "Software engineering of machine learning systems," *Communications of the ACM*, vol. 66, no. 2, pp. 35–37, 2023.
[77] S. Cao, X. Sun, R. Widyasari, D. Lo, X. Wu, L. Bo, J. Zhang, B. Li, W. Liu, D. Wu *et al.*, "A systematic literature review on explainability for machine/deep learning-based software engineering research," *arXiv preprint arXiv:2401.14617*, 2024.
[78] S. Wang, L. Huang, A. Gao, J. Ge, T. Zhang, H. Feng, I. Satyarth, M. Li, H. Zhang, and V. Ng, "Machine/deep learning for software engineering: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 49, no. 3, pp. 1188–1231, 2022.
[79] A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, S. A. Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym *et al.*, "Developments in mlflow: A system to accelerate the machine learning lifecycle," in *Proceedings of the fourth international workshop on data management for end-to-end machine learning*, 2020, pp. 1–4.
[80] C. Weber and P. Reimann, "Mmp - a platform to manage machine learning models in industry 4.0 environments," in *2020 IEEE 24th International Enterprise Distributed Object Computing Workshop (EDOCW)*, 2020, pp. 91–94.
[81] D. OBrien, S. Biswas, S. Imtiaz, R. Abdalkareem, E. Shihab, and H. Rajan, "23 shades of self-admitted technical debt: An empirical study on machine learning software," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 734–746.

[82] S. Sztwiertnia, M. Grübel, A. Chouchane, D. Sokolowski, K. Narasimhan, and M. Mezini, "Impact of programming languages on machine learning bugs," in *Proceedings of the 1st ACM International Workshop on AI and Software Testing/Analysis*, 2021, pp. 9–12.